Static Scheduling with Interruption Costs for Computer Vision Applications

Francisco A. Candelas, Fernando Torres, Pablo Gil, Santiago T. Puente

DFISTS, University of Alicante,
Campus of San Vicente del Raspeig, P. O. Box. 99, E-03080 Alicante, Spain
{fcandela,medina,pgil,spuente}@disc.ua.es
Phone & Fax: +34 96 590 36 82

Abstract. It is very difficult to find pre-emptive scheduling algorithms that consider all the main characteristics of computer vision systems. Moreover, there is no generic algorithm that considers interruption costs for such systems. Taking the interruption of tasks into account scheduling results can be improved. But it is also very important to take the costs that arise from interruptions into account because they not only increase the total execution time, but also because the scheduler can evaluate whether it is adequate to interrupt certain tasks or not. Thus, the result can be more realistic. Therefore, we present an extension to the static algorithm SASEPA for computer vision which considers interruption costs.

Keywords. Static scheduling, interruption costs, computer vision, task.

1 Introduction

Of the many specific characteristics of computer vision applications regarding scheduling, we would like to emphasize the following:

1. Multi-processor systems with different kinds of processors are generally employed. Thus, techniques for spatial allocation and temporal scheduling which consider it are needed. It is possible to differentiate between generic *CPUs* and *IAPBs* (Image Acquiring and Processing Boards). This classification involves three basic kinds of tasks: *cpu, iapb* and *cpu/iapb*. The last one is a communication task between processors of different kinds [1][2].
2. There are precedence and exclusion relations among tasks. The first ones are determined by the executing order and the data flow between the operations. The second ones establish what tasks can be interrupted by others and which can not be, and they are considered in pre-emptive scheduling techniques [3][4].
3. The elementary tasks are generally sporadic, and their creation times depends on precedence relations.

Taking task interruptions into account, allow more flexibility in distributing tasks in scheduling [3]. This can improve the results since the execution time of the processors is better used and the parallelism of the execution is greater [4][5].

Though many scheduling strategies are been developed and implemented, few are directly designed for computer vision applications. Moreover the majority are designed for specific architectures and they do not take all of the important characteristics of such applications into account [1][2]. As an example, we can consider techniques such as the PREC 1 [6], the Empty-Slots Method [7] or the Critical Path [1][2]. The first one takes the precedence relations, sporadic tasks and interruptions into account, but it does not consider different kinds of processors. The second one considers the different characteristics, but it works with sporadic tasks and is designed for RT-LANs. The third takes all of the characteristics mentioned above into account, but it does not make spatial allocation for a multi-processor system. Also, there are scheduling strategies for computer vision as the one proposed in [8] and [9], but this does not consider interruptions or different kinds of processors.

References [10] and [2] describe the static algorithm SASEPA (Simultaneous Allocation and Scheduling with Exclusion and Precedence Relations Algorithm), which carries out a spatial allocation and a temporal scheduling over a multi-processor system considering all of the above-mentioned characteristics for computer vision systems. This algorithm also does a pre-emptive scheduling and considers the task interruptions to make the resulting scheduling better. But it does not take the temporal costs derived from interruptions into account. Because it is static, it is suitable for the research and design steps of an computer vision application [3].

In this paper, a SASEPA extension that considers interruption costs is proposed. It represents a new approach in the scheduling algorithms for computer vision applications which can be also applied to other systems. After describing the basic aspects of SASEPA below, Section 2 explains how interruptions and their costs are modelled. Next, Section 3 describes how interruption costs are considered in the SASEPA extension. The practical evaluation of the proposed extension is described in Section 4. Finally, our conclusions are presented in Section 5.

1.1 Basic Aspects of SASEPA

Each high-level operation of a computer vision application can be divided into a set of elementary tasks. The scheduling algorithm takes a DAG (Directed Acyclic Graph) which contains the attributes of the tasks and the relations between them as an input [6]:

$$G=(T, A) \; . \qquad (1)$$

$$T=\{\tau_1, \tau_2, \ldots, \tau_N\}, \;\; \tau_i=(c_i, k_i, \mathrm{IntCost}_i), \;\; i=1,2,\ldots,N \; .$$

$$A=\{(\tau_i, \tau_j) \,/\, \tau_i \text{ precedes } \tau_j\}, \;\; i,j=1,2,\ldots,N \; .$$

Each task has its computation time c_i and its kind k_i (which can be *cpu*, *iapb* or *cpu/iapb*) associated. It also has a set o function $IntCost_i$ that gives interruption costs. The creation time of each task is not explicitly specified and it is determined by means of the precedence relations. Nor are the deadlines considered.

The result of the algorithm is the spatial allocation and the temporal scheduling of tasks in the available processors (CPU and IAPB) taking the kinds of tasks and the relations between they into account. The algorithm also minimizes the total execution time, making the necessary interruptions of tasks to do so.

Fig. 1 shows the main steps of the SASEPA. After initiating the algorithm, ready tasks are searched for among the unfinished tasks. These are tasks that have all their preceding tasks finished and the minimum creation time. The creation time of a task depends on the finishing time of its preceding tasks and the accumulated delay due to interruptions and deferments of the task due to a lack of free processors.

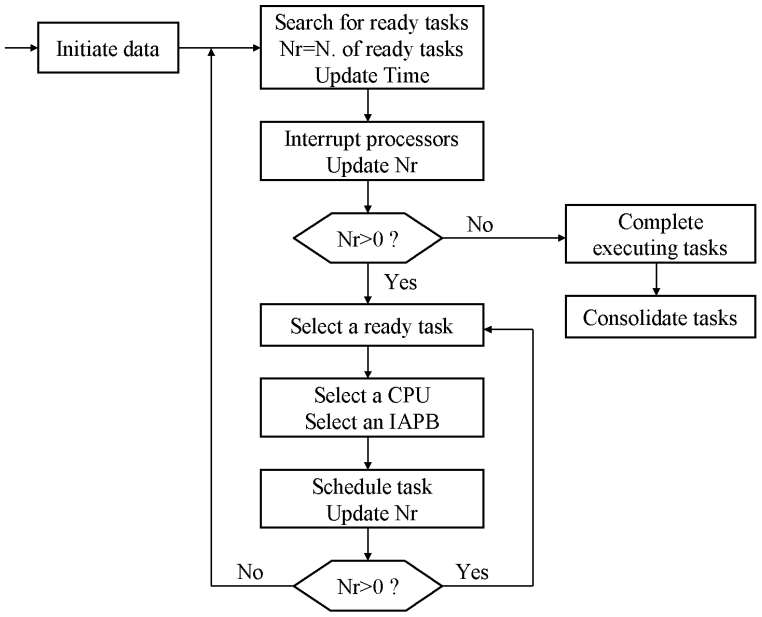


Fig. 1. Block diagram of the main procedure of SAPEPA

Next, all the processors executing tasks that can be interrupted are interrupted, and corresponding tasks also become ready tasks. The interruption costs are not considered.

If there are ready tasks (*Nr* > 0), one is selected to be scheduled. The criterion for selecting a task is based on finding the *critical task* first, which is the task with a maximum finishing time. Then a ready task that is a predecessor of the critical task is selected. In order to resolve ties among several tasks, a weight is associated to each one, and the task with the highest weight is selected. The weight of a task expresses the current computation time required to execute it and all of its successors with the maximum parallelism. Thus, the selected task is the one that delays the total execution most.

Then, a the suitable processor (or processors if the task is *cpu/iapb*) is chosen for the selected task and this is scheduled. The selection is based on minimizing communications between different processors.

When a task is scheduled, one of two situations may occur. If the processor (or processors) is free, the task is scheduled in it, and the number of ready tasks (*Nr*) is decreased. On the other hand, if the processor is not free (it executes a non-interruptible task; otherwise, the processor would have been interrupted before) the task is not scheduled to test the selection of another processor in a future iteration of the internal loop. When all of the processors are tested without success, the task is delayed and *Nr* is decreased.

The above steps are repeated while there are ready tasks to be scheduled. When it is no longer possible to find ready tasks, the loop ends. Then the tasks that are still being executed are completed, and finally, all the tasks are consolidated. This last step removes unnecessary interruptions made by the algorithm.

2 Interruption Costs

In computer vision systems, as in others, it is necessary to save the state of a task when it is interrupted, to be able to resume its execution in the future. It is also necessary to retrieve the saved state just before resuming the task. These operations, which are called context switching, involve time costs which may become important if many interrupts are generated. Because of this, it is advisable to bear these costs in mind. Moreover, if the scheduler takes the interruption costs into account it can evaluate whether it is suitable to make an interruption or not.

Fig. 2 shows how the interruptions affect any task t_i. Due to the interruptions, the execution of a task may be broken down into several intervals of time. If a generic interval j is considered, with $j \in \{1,2,...,M\}$, a reading cost $r_{i,j-1}$ is required to retrieve the original state of the task at the beginning of the interval. The cost $r_{i,j-1}$ depends on the previous interruption. Furthermore, a writing cost $w_{i,j}$ is required to save its the state at the end of the interval. For the first interval of a task $r_{i,0}$ is 0, and for the last interval $w_{i,M}$ is 0.

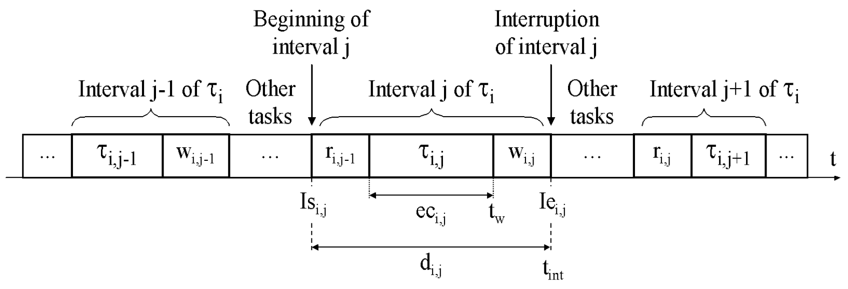


Fig. 2. A task broken down into several execution intervals, due t o the interruptions

Because of the interruption costs, the effective time spent in computing a task ($ec_{i,j}$) is shorter than the duration of the interval ($d_{i,j}$), and the finishing time of the task is postponed. Thus, if c'_i is defined as the remaining time of computation of the task t_i, this value is increased in each interruption of an interval j according to $w_{i,j}$+ $r_{i,j}$:

$$c'_{i,0} = c_i \; . \tag{2}$$

$$c'_{i,j} = c'_{i,j-1} - d_{i,j} + w_{i,j} + r_{i,j} \quad j=1,2,...,M \; .$$

It is noteworthy that the writing of the state is performed within the corresponding interval before the instant of interruption. That is, given a desired instant of interruption t_{int}, the writing of state $w_{i,j}$ is considered just before this instant, beginning at t_w. In this way the scheduler can get the desired length of time $d_{i,j}$ for the interval. This approach simplifies the interruption management. However, it is necessary that the scheduler algorithm is static to be able to carry it out.

To simplify the modelling and the management of interruptions by the scheduler, the costs of writing and reading the state can be considered constant for each task t_i:

$$w_{i,1} = w_{i,2} = \ldots = w_i \; . \tag{3}$$

$$r_{i,1} = r_{i,2} = \ldots = r_i \; .$$

2.1 Interruption Cost Function

However, costs for writing and restoring the state of a task are not constant, but depend on the instant of time t_w in which the task interruption begins. This instant is measured relative to the effective computation time of the task. For example, let us consider an operation for computer vision that searches for some characteristics of an image and processes them all at the end. The more advanced the operation is, the more information about characteristics detected will have to be saved temporarily in case of interruption.

Thus, a more realistic but more complex model is considering a function for each task that returns the writing and reading costs for it. The parameter of these functions is the instant in which the interruption begins in relation to the effective computation time of the task:

$$(w_i, r_i) = IntCost_i(t_w) \; , \; t_w \in [0, c_i) \; . \tag{4}$$

This cost function can be defined by the different intervals of time that involve different writing and reading costs. As an example, let us consider the function that Fig. 3 shows, which can be expressed in this way:

$$(w_i, r_i) = IntCost_i(t_w) = \begin{cases} (1,1) & 0 \le t_w < 2 \\ (2,1) & 2 \le t_w < 5 \\ (2,3) & 5 \le t_w < 11 \\ (4,6) & 11 \le t_w < 14 \end{cases} \; . \tag{5}$$

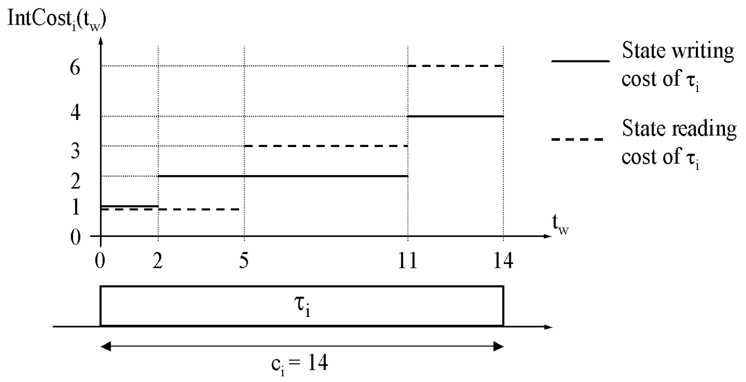


Fig. 3. Example of a cost function for a task t_i

2.2 Considerations about the Interruption Cost Function

When the cost function is used to determine interruption costs in a static scheduling, two problematic situations may arise. To illustrate the first situation, let us suppose that the scheduler needs to interrupt the first interval of a task t_i in instant t_{int}=10. The task has the following cost function associated:

$$\mathrm{IntCost}_i(t_w) = \begin{cases} (9,3) & 0 \le t_w < 6 \qquad (2,2) \quad 8 \le t_w < 9 \\ (4,3) & 6 \le t_w < 7 \qquad (3,4) \quad 9 \le t_w < 10 \\ (3,4) & 7 \le t_w < 8 \qquad (4,3) \quad 10 \le t_w < 12 \end{cases} . \tag{6}$$

For the sake of simplicity, the interval starts at instant 0 of time. The scheduler disposes of the following options to carry out that interruption: to initiate the state writing at t_w=8 which involves a writing cost $w_{i,1}$=2; to initiate state writing at t_w=7 with $w_{i,1}$=3; or to consider t_w=6 with $w_{i,1}$=4. These cases are illustrated in Fig 4.

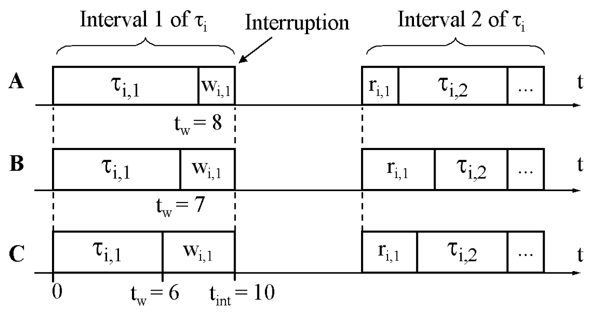


Fig. 4. Task with three interruption options for instant t_{int}=10

In the previous example, the best option is *A* since it maximizes the effective computation time of the task for the interrupted interval.

Now let us consider a new cost function for t_i:

$$\mathrm{IntCost}_i(t_w) = \begin{cases} (4,3) & 0 \le t_w < 5 \\ (5,4) & 5 \le t_w < 8 \\ (3,2) & 8 \le t_w < 12 \end{cases} . \qquad (7)$$

In this case, if the scheduler wants to interrupt the first interval at t_w=10 there are no possible options to finish the interval at that precise instant. The best option is to begin the writing at t_w=4 and finish the interval at t_{int}=8 as shown in Fig. 5.

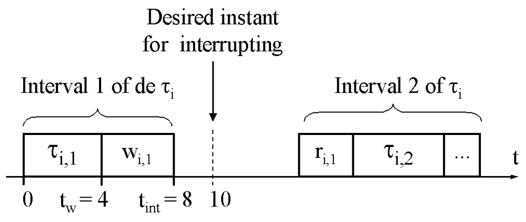


Fig. 5. Interval which can not be interrupted at the desired instant

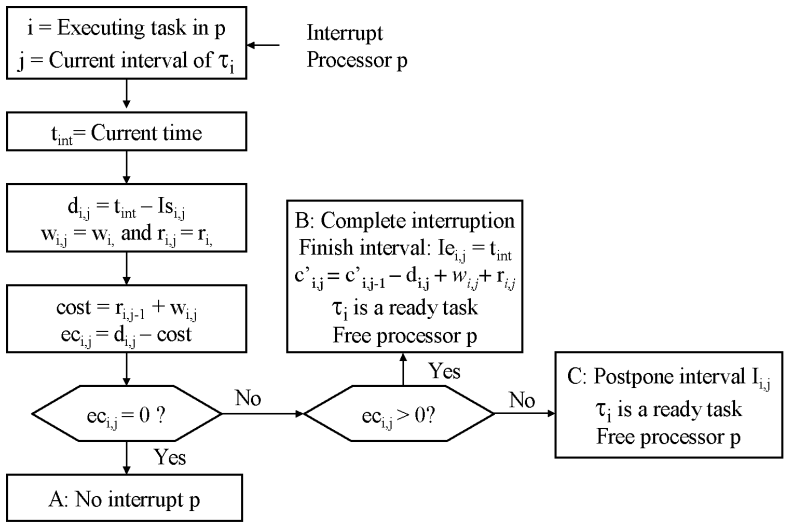
The criterion that the scheduler must apply to solve the former situations when it needs to interrupt a task is not just to determinate the instant t_w which involves a interval that finishes before the desired instant of interruption, but also to maximize the effective computation time for the task.

3 SASEPA with Interruption Costs

We have developed an extension of the SASEPA algorithm explained in Section 1.1. This extension considers the aspects related to the task interruptions that were described in Section 2. As described in that section, the interruption costs must be considered when a task is being scheduled or interrupted. Thus, these two operations will be the next procedures to be described.

3.1. Interruption of a task

Fig. 6 shows the steps required to interrupt a processor, considering that interruption costs are constant, as (3) expresses. Three different situations can be distinguished depending on the duration of the interval that has been interrupted in relation to the costs of the interruptions. In case A, the interval is just long enough to include the costs of the interruption, but not the effective computation time. In such a case, the execution of the task is allowed to continue. In case C, the interval is pot long enough to include the effective computation of the task, and so it is temporarily postponed.



i = Executing task in p
j = Current interval of τ_i
Interrupt Processor p
t_{int}= Current time
$d_{i,j} = t_{int} - Is_{i,j}$
$w_{i,j} = w_i$, and $r_{i,j} = r_i$,
$cost = r_{i,j-1} + w_{i,j}$
$ec_{i,j} = d_{i,j} - cost$
$ec_{i,j} = 0$?
No
Yes
A: No interrupt p
$ec_{i,j} > 0$?
Yes
No
B: Complete interruption
Finish interval: $Ie_{i,j} = t_{int}$
$c'_{i,j} = c'_{i,j-1} - d_{i,j} + w_{i,j} + r_{i,j}$
τ_i is a ready task
Free processor p
C: Postpone interval $I_{i,j}$
τ_i is a ready task
Free processor p

Fig. 6. Steps followed to interrupt a task

The same steps are followed in quite a similar way to consider the cost functions where they are specified. The only difference is that now it is necessary to calculate the instant t_w by means of the cost function of a task before calculating t_{int}. If t_w is found, then $r_{i,j}$, $w_{i,j}$, t_{int} and $d_{i,j}$ are determined from it. On the other hand, if t_w is not found, the interval is postponed. Furthermore, it is necessary to time the effective duration of the computation of each task to be able apply the cost functions.

3.2 Scheduling a task

The procedure shown in Fig. 7 is followed to schedule a pre-emptable task, considering constant interruption costs. If the assigned processor is free then one of two basic situations can occur: either the previous interval can be continued or it is necessary to start a new interval, depending on how and where the previous interval of the task was finished.

If the previous interval was interrupted in the same processor just before the instant which is being scheduled, it is possible to continue that interval (case A). In this case, the interruption costs that were considered before must be subtracted from $c_{i,j}$. If the previous interval was postponed in the same processor it is also possible to continue it, to achieve a longer interval (case C). In this case, it is not necessary to subtract the interruption costs, since they were not considered before. In other cases, a new interval must be considered.

The same steps are considered to take the cost functions into account during scheduling, but adding a new feature: it must be possible to continue an interval which has been interrupted before the desired instant.

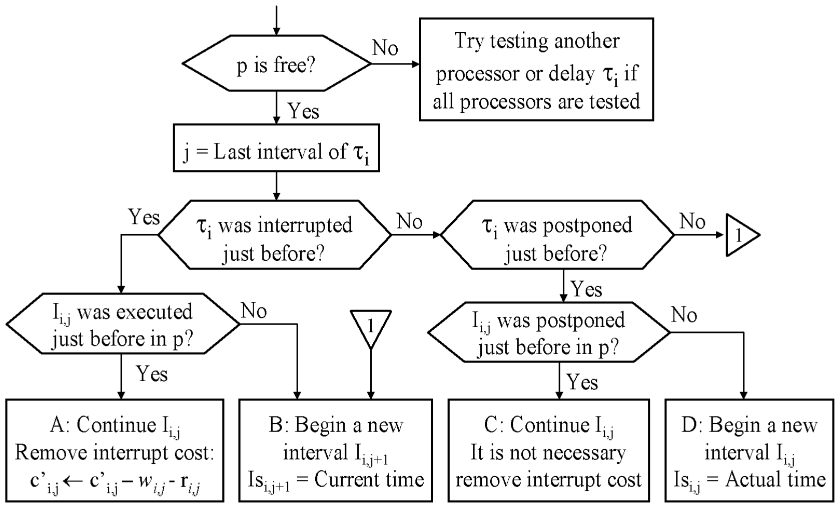

p is free?
No
Try testing another processor or delay τ_i if all processors are tested
Yes
j = Last interval of τ_i
Yes
τ_i was interrupted just before?
No
τ_i was postponed just before?
No
1
$I_{i,j}$ was executed just before in p?
No
1
Yes
$I_{i,j}$ was postponed just before in p?
No
Yes
Yes
A: Continue $I_{i,j}$ Remove interrupt cost: $c'_{i,j} \leftarrow c'_{i,j} - w_{i,j} - r_{i,j}$
B: Begin a new interval $I_{i,j+1}$ $Is_{i,j+1}$ = Current time
C: Continue $I_{i,j}$ It is not necessary remove interrupt cost
D: Begin a new interval $I_{i,j}$ $Is_{i,j}$ = Actual time

Fig. 7. Step followed to schedule (begin or continue) a task

4. Evaluation

To evaluate the proposed SASEPA extension that considers interruption costs, a real computer vision application has been considered; a correspondence algorithm for the characteristics of two images captured with a pair of stereoscopic cameras.

The first step was to define the tasks and to estimate their characteristics, including state writing and reading costs for each task. We should point out that the developed extension can manage both constant costs and function costs models for each task, and that the two models can be used in the very execution of the algorithm. The cost function has only been defined for the more complex tasks.

Afterwards, the application was specified as a high-level scheme using the tools described in [2] and [11]. These tools also generate the task graph that has been used as the input for the static scheduling algorithms tested. Fig 3. shows this DAG. Table 1 shows the main characteristics of the most outstanding tasks of the task graph which will be discussed later on.

Table 1. Characteristics of the most outstanding tasks of the graph in Fig. 8

Task	Kind	Execution time	Writing costs	Reading costs
5	iapb	118 ms	5 ms	5 ms
23	iapb	130 ms	5 ms	5 ms
31	iapb	130 ms	5 ms	5 ms
32	iapb	130 ms	5 ms	5 ms
93	cpu	600 ms	$IntCost_{93}()$	

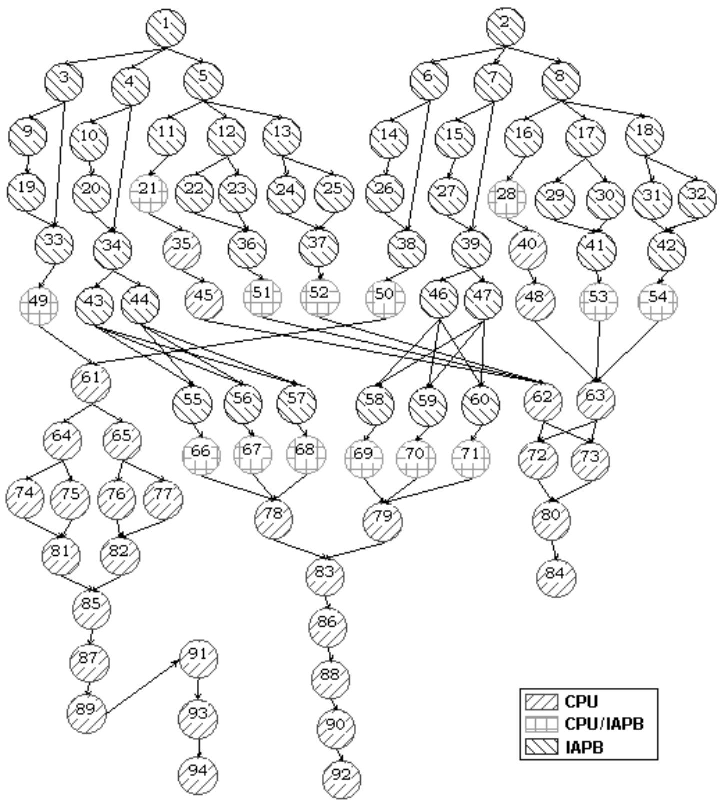


Fig. 8. Tasks graph used to evaluate the scheduling algorithms

The tasks have been scheduled using four different scheduling algorithms considering a target architecture with a CPU and two IAPBs. The four algorithms were the PREC 1 [6], the Critic Path [1], the SASEPA [10] and the SASEPA extension with interruption costs. In order to apply the two first algorithms for the target architecture it was necessary to improve then with several new features (kinds of tasks, sporadic tasks, spatial allocation…).

The main results for the four algorithm are shown in Table 2. We can see that the SASEPA executes all of the tasks in less time and with a higher processor occupation than the PREC 1 and the Critic Path. Moreover, the SASEPA makes fewer interruptions. Regarding the SASEPA extension, it takes more time to execute all of the tasks and decreases the processor occupation. This it is logical because it takes the interruption costs into account. Furthermore, the SASEPA extension interrupts different tasks than the previous algorithm. This shows how this algorithm considers interruption costs to decide what tasks it can interrupt. This important aspect is explained in more detail below.

Table 2. Result of scheduler algorithms for the graph in Fig. 8

Scheduler	Execution time	Processor occupation	Number of interruptions	Interrupted tasks
PREC1 - M	2494 ms	56 %	0	-
Critic Path - M	1774 ms	79 %	8	43, 46, 83 (3), 93 (3)
SASEPA	1766 ms	79 %	5	5, 31 (2), 32 (2)
SASEPA with int. costs	1838 ms	77 %	5	23, 93 (4)

The resulting scheduling of the SASEPA is shown in Fig. 9, and Table 3 details the intervals into which the interrupted tasks are broke down. We can verify that intervals $I_{5,1}$, $I_{31,1}$ y $I_{32,2}$ are not long enough to be able execute the state reading and writing in accordance with the values shown in Table 1. Some intervals are even just one or two milliseconds long, in contrast with the total duration of over a hundred milliseconds of the task. As such, they are invalid intervals for an implementation in practice.

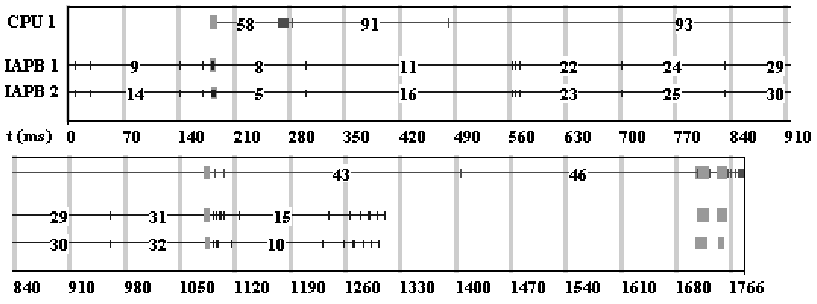


Fig. 9. Resulting scheduling of SASEPA

Table 3. Intervals of tasks interrupted by SASEPA

Task	Intervals (ms)	Length of intervals (ms)	Total length (ms)
5	(182,184), (186,302)	2, 116	118
31	(96, 1083), (1084,1092), (1100,1101)	121, 8, 1	130
32	(96, 1084), (1085,1092), (1096,1097)	123, 7, 1	130

In contrast, Fig. 10 shows how the resulting scheduling, when the SASEPA extension is considered, is different from the result shown in Fig. 9, previously commented. This is because this algorithm has decided to interrupt other tasks, which have been broken down into the intervals detailed in Table 4. In this case, the intervals are sufficiently long to execute the context switching, in addition to a portion of the task. Thus, it is possible to implement the resulting scheduling in practice.

Table 4 also shows how the durations of the tasks are increased by including the interruption costs.

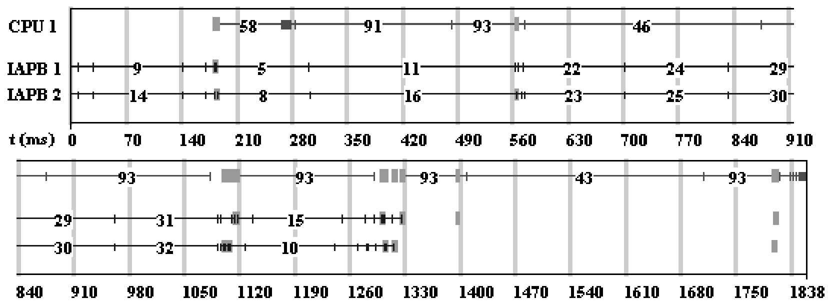


Fig. 10. Resulting scheduling of SASEPA with interruption costs extension

Table 4. Intervals of tasks interrupted by SASEPA with interruption costs

Task	Intervals (ms)	Length of intervals (ms)	Total length (ms)
23	(575,702), (1096,1109)	127, 13	130+10
93	(483,562), (875,1083), (1117,1290), (1326,1396), (1707,1797)	479, 208, 173, 70, 90	600+420

It must be remembered that the of scheduling algorithm employed is static, and the scheduling of the tasks is done in an off-line manner before they are executed. Even the system in which the scheduler is executed can be different from the target system. As such, the cost of scheduling does not influence the final execution. Furthermore, the algorithm has been originally designed for computer vision systems in which the static scheduling is done in the first stages of the design, and in this case, the costs in time and space are not much more important than other aspects like the friendliness that the interface of the design tools should offer. As such, we have not considered a detailed study of the costs or a comparison with other algorithms necessary. In any case, other static scheduling algorithms with the same features as the one we propose here do not exists and, as such, a direct comparison would not be useful.

However, we have verified, in practice, that the SASEPA static scheduler with the model presented here has a lower time cost. So much so, that a dynamic version of the scheduler is now being studied. For example, with an implementation of the algorithm for MS Windows 95 being used in a Pentium III of 450MHz PC, a complete scheduling of the computer vision application mentioned in Section 4 can be done in 170ms.

5. Conclusions

In this paper we have presented a new model which allows us to consider the costs involved in reading and writings the state, derived from interruptions made by a static scheduling algorithm. In this particular case, an SASEPA extension for computer vision applications has been considered. However, the model can be applied to other pre-emptable static scheduling algorithms.

Interruption costs can be modelled as constant values or as a cost function for each task. Although the second approach is more realistic, it is usually difficult to estimate such functions for all tasks in practice, and it is easier to considerer constant costs. However, the extension developed allows us to use the two models simultaneously, choosing the most suitable one to express the costs over tasks, according to the characteristics of the tasks.

The proposed model is interesting for other applications that use static and pre-emptive scheduling, since it offers a more realistic result, as it does not create any interruptions that are impossible to carry out later on in practice. However, to be able to apply the model directly, the costs of task interruptions must be known or estimated some how. This way, the model is useful in applications whose task are well defined before their scheduling, such as computer vision applications like the industrial inspection of products, in which the characteristics of the tasks to be done and the images to be processed are known in advance. In other words, the task algorithms and how their execution depends on the images that they process are known. This way, the costs of storing and retrieving the state of the tasks can be estimated, even in relation to the part of the task that has already been carried out at a given moment.

When interruption costs are considered in a pre-emptable static scheduling algorithm, not only is a more realistic and generally longer execution time obtained, but also the tasks can be scheduled in a more intelligent way. In other words, the scheduler can avoid interruptions that can not be implemented in practice or that are not the most appropriate.

We should point out that it is not necessary to know the exact costs of interruption to enjoy the advantages of the model proposed here. By estimating the values of the costs of interruption concerning the execution of the task and taking them into account in the scheduling, a certain “intelligence” can be afforded to the pre-emptive scheduler so that it can decide whether it is more convenient to interrupt a certain task at a given moment or not.

References

1. Torres, F., Candelas, F.A., Puente, S.T., Jiménez, L.M. et al.: Simulation and Scheduling of Real-Time Computer Vision Algorithms. Lecture Notes In Computer Science, Vol. 1542. Springer-Verlag, Germany (1999) 98-114
2. Candelas, F.A.: Extensión de Técnicas de Planificación Espacio-Temporal a Sistemas de Visión por Computador. P.D. Thesis. University of Alicante, Spain (2001)
3. Nissanke, N.: Realtime Systems. Prentice Hall Europe, Hertfordshire (1997)
4. Zhao. W, Ramamritham, K., Stankovic, J.A.: Preemptive scheduling under time and resource constrains. IEEE Transactions on Computers, Vol. 36 (1987) 949-960

5. Xu, J., Parnas, D.L.: On satisfying timing constrains in hard-real-time systems. IEEE Transactions on Software Engineering, Vol. 19 (1993) 74-80
6. Krishna, C. M., Shin, K. G. : Real-Time Systems. McGraw-Hill (1997)
7. Santos, J., Ferro, E., Orozco, J., Cayssials, R.: A Heuristic Approach to the Multitask-Multiprocessor Assignment Problem using Empty-Slots Method and Rate Monotonic Scheduling. Real-time Systems, Vol. 13. Kluwer Academic Publishers, Boston (1997) 167-199
8. Lee, C., Wang, Y.-F., Yang, T.: Static Global Scheduling for Optimal Computer Vision and Image Processing Operations on Distributed-Memory Multiprocessors. Technical Report TRC94-23, University of California, Santa Barbara, California (1994)
9. Lee, C., Wang, Y.-F., Yang, T.: Global Optimization for Mapping Parallel Image Processing Task on Distributed Memory Machines. Journal of Parallel & Distributed Computing, Vol. 45. Academic Press, Orlando, Fla. (1997) 29-45
10. Fernández, C., Torres, F., Puente, S.T.: SASEPA: Simultaneous Allocation and Scheduling with Exclusion and Precedence Relations Algorithm. Proc. PPAM'2001, Naleczow, Poland (2001)
11. Torres, F., Candelas, F.A., Puente, S.T. Ortíz, F.G.: Graph Models Applied to Specification, Simulation, Allocation and Scheduling of Real-Time Computer Vision Applications. International Journal of Imaging Systems & Technology, Vol. 11. John Wiley & Sons, Inc., USA (2000) 287-291