# Scheduling Hard Real-Time Tasks in Multicore General Purpose/ Special Purpose Processors Systems-on-a-Chip: An Energy-Aware Approach

Rodrigo Santos[1], Jorge Santos[1], Javier Orozco[1], David Donari[1], Leo Ordinez[1]

**Abstract** – *In this paper an energy-aware scheduling algorithm for heterogeneous multicore general purpose/special purpose processors systems-on-a-chip is presented. The load consists in chains of precedence-related tasks. A systematic search method to find an optimal set of reduced frequencies that diminish the energy consumption is proposed. Evaluations were conducted by means of extensive simulations.*

***Keywords**: Real-Time, Multicore, Embedded Systems*

## I. Introduction

In the last years the computer industry trend has changed from very fast uniprocessor systems working over the Gigahertz domain to a set of assembled processors working at a lower frequency. Multicore systems are becoming more common in the industry as performance and energy consumption placed a limit in the development of new devices. The possibility of having a set of tight connected processors with less individual computational power but with a greater assembled one, has produced a new trend in the design of computers.

In heterogeneous single-chip multicore devices, portions of the chip can be differentiated at a high level of abstraction to implement more efficiently the required applications within the restricted area of the chip [1]. In the paper of reference, the author poses a set of questions about this kind of systems, one of which is how to schedule across multiple heterogeneous cores.

This paper intends to give one answer to that question for the case of Systems-on-a-Chip, SoC's, composed of a General Purpose Processor (GPP) and one or more Special Purpose Processors (SPP) connected in a virtual star topology with the GPP at its hub. This is the trend presently followed by the industry, exemplified by the Texas Instruments SMJ320C80 [2] (one GPP, four digital signal processors, DSP's, and two controllers), the Texas Instruments TMS320DM6443 [3] (one ARM926 RISC processor, one DSP plus audio-video accelerators), the Aplio/TRIO [4] (one ARM7TDMI ARM Thumb Processor Core with two 16-bit Fixed-point Oak DSP Core), and the IBM Cell Broadband Engine Architecture [5] (one GPP and eight media processors). Although some tasks may be completely executed in the GPP, most of them will be mainly processed in some SPP acting as accelerator of the GPP, which, in turn, acts as a master processor distributing tasks to the SPPs. In this way, chains of real-time precedence-related tasks, executed in different processors, are formed.

In the last years, the industry has shifted from maximizing performance to maximizing performance per watt [5]. The energy consumption increases by a multiplicative factor as the speed of a single processor is increased, but only by an additive factor as processors are added [6]. This is the rationale for using multiple simple cores instead of single but complex processors [7] and explains the actual multicore trend leading to the kind of SoC's studied in this paper.

Since for a given architecture, additional savings can be obtained by a proper handling of the available slack, the subjects addressed in this paper are not only to schedule sets of precedence-related real-time tasks to be executed in multicore heterogeneous systems but also how to save energy after a proper assignment meeting all time-constraints has been made. In order to do it, a schedulability test for each processor and the computation of a suboptimal single frequency, selected from a discrete set of available frequencies, are carried out. However it must be borne in mind that since tasks are related by precedence, frequencies in the processors are also related and cannot be selected independently.

### *I.1. Contribution*

This paper presents an energy-aware scheduling algorithm for heterogeneous multicore SoC. The mechanism proposed a necessary and sufficient scheduling test for precedence related chains of tasks that execute in a sequential way in the different processors. Although the algorithm presented contemplates three execution stages it can be easily extended for more resources.

### I.2. Organization

The rest of the paper is organized as follows: In Section 2 previous works on scheduling and energy saving in single and multiprocessor systems are analysed. In Section 3 the system model is discussed. The non-preemptive method for scheduling precedence-related tasks, and the method to save energy are presented in Sections 4 and 5, respectively. In Section 6 experimental results based on simulations are shown and finally in Section 7, conclusions are drawn and future work outlined.

## II. Previous Work

The list of the analyzed papers is representative of the state of the art. The problem of scheduling multiprocessors has been addressed in many papers. In [6]}, the models differ from the one used in this paper in that processors are homogeneous (also called symmetric) and/or tasks are independent. In [8] Rajkumar proposed the Distributed Priority Ceiling Protocol to schedule tasks to be executed in a one-GPP/one-SPP system working under a Fixed Priority discipline. Although the method could be extended to one-GPP/several-SPP's systems, it has the problem (analysed in [9]) that blockings in the SPP's are always taken into consideration, even for tasks that execute only in the GPP. Because of this, some feasible systems would be deemed to be non-schedulable by this method.

In [9], the proposal of Rajkumar is improved by assembling two scheduling queues, one for chains executed at the DSP and the other one for tasks executed only at the GPP. However, the DSP is not considered to be an independent processor but it is seen as a critical section of the GPP. This lowers the utilization factor of the system to be processed, an inconvenience saved in the scheduling method proposed in the next section. Moreover, both approaches use a Fixed Priority discipline instead of the dynamic Earliest Deadline First, EDF, of the system model described in Section 3

In [10] two scheduling methods for the synchronization of processes in distributed systems are presented. The time that a process is waiting for a response from another process is called *external blocking*. A scheduling algorithm based on the Response Time Analysis [11] divides the tasks in two independent parts, before and after the external blocking. While the first part has a deterministic release time, the second one has a release jitter which is equal to the external blocking. Although the proposition provides an interesting way to analyze the feasibility of the system, the schedulability of the second processor is not discussed. A Fixed Priority discipline is used.

In [12] the authors describe a Dynamic Voltage Scheduling algorithm, DVS, for saving energy in non-preemptive systems working under EDF. The method is restricted to only one processor and is dynamic while in this paper the scaling is static.

In [13], a variant of non-preemptive EDF is proposed for soft real-time systems. The proposed algorithm, named gEDF, groups tasks with near deadlines and schedules them following the Shortest Job First (SJF) policy. The authors show that in the case of uniprocessors, the performance under overload is better than the traditional EDF approach.

## III. System Model

This paper deals with sets of precedence related tasks. $\gamma_i$ and $\tau_{ij}$ denote the $i^{\text{th}}$ set and its $j^{\text{th}}$ task, respectively. Two tasks are said to be related by precedence when, in order to be executed, a task $\tau_{ij}$ needs data produced by other task, $\tau_{ig}$. The relation, notated $\tau_{ig} \prec \tau_{ij}$, determines a partial ordering of the tasks. When there is no task $\tau_{ih}$ such that $\tau_{ig} \prec \tau_{ih} \prec \tau_{ij}$, $\tau_{ig}$ and $\tau_{ij}$ are called predecessor and successor, respectively. In what follows it will be assumed that all the tasks in the set have the same period. Although it may be the case that different tasks in the set can have different periods, the analysis here performed considers the minimum one as the period of the set. This imposes an extra restriction that may be relaxed later if necessary. $C_{ij}$, $T_{ij}$ and $D_{ij}$ shall denote the worst-case execution time, the period and the deadline of $\tau_{ij}$, respectively. Different tasks may execute in different processors running at different speeds.

Scheduling disciplines in the different processors are determined by the kind of tasks executed in them. For example, the SPP's generally require a non-preemptive discipline because they perform specific functions based on registers values. The context-switch, associated to the preemption, imposes a cost that is too high and diminishes the advantages that a preemptive discipline can have. Instead, for the case of the GPP, a preemptive discipline provides a better utilization of the processor.

The existence of shared resources is not particularly considered in this paper. For the case of the individual SPPs, as the scheduling discipline is not preemptable, there is no need for a contention algorithm. The possibility of introducing shared resources in the GPP or even between the different processors may be contemplated by means of the blocking relation presented in [8].

## IV. Heterogeneous Real-Time Schedulability

In this section, the problem of scheduling sets of tasks consisting in chains of precedence-related periodic real-time tasks as defined in Section III is addressed. The basic idea of the method here presented is to convert the precedence-related tasks in sets of independent periodic tasks, with their own release times and deadlines, to be executed in heterogeneous processors.

Several priority disciplines have been proposed to be used in the scheduling of real-time systems. In preemptive EDF, a higher priority task gains access to the processor for the duration of an execution slot, by displacing a lower priority task. The status of the preempted task, mainly registers' contents, must be preserved in order to resume its execution at a later time. In [14], it is proved that preemptive EDF is optimal if the time required to switch from one task to another one is neglected. However, if switching times are taken into consideration and they are expensive, non-preemptive EDF may turn to be better. This is the reason why preemptive EDF is used in the GPP while non-preemptive EDF is used in the SPP's.

In [15] a method, using preemptive EDF, was presented to schedule single processor real-time systems composed of precedence-related tasks. It can therefore be applied to our GPP scheduling; in order to do it, the chain of precedence-related tasks must be converted into a set of independent tasks. Once this is done, the tasks are allocated to the different processors in such a way that time-constraints are met; following that, a reduction in energy consumption may be attempted.

The central idea to be used is the Chetto definition of a coherent system [15]. In it, each task has its own release time and deadline. The release time, deadline and period of the set of precedence related tasks, $\gamma_i$, are notated $r_i$, $D_i$ and $T_i$, respectively. The release and deadline of $\tau_{ij}$, notated $r_{ij}$ and $d_{ij}$ respectively, should be selected in such a way that the processors are coherent. A processor is coherent if the release of a succesor task executing in it is greater than or equal to the deadline of its predecessor in the chain. Therefore, $r_{ij} < d_{ij} \leq r_{ik} < d_{ik}$ must hold. In this paper the number of tasks considered in a chain is at maximum three. The first and the last one run in the GPP while the second one executes in one of the SPP's. The system is feasible if all the processors are schedulable.

In the SPP's, tasks are independent among them so the schedulability can be analysed following the test proposed in [16] for non-preemptive EDF.

## V. SPP's Schedulability

To check the schedulability of the SPP's and the GPP, release times and deadlines for each task in the system should be computed; this is not an easy job because they are not independent. In order to begin the analysis, a preliminary value for the deadline of each of the SPP's tasks has to be set. If the task has a successor in the GPP, then the preliminary value will be the latest possible release of the successor. This, however, does not secure the schedulability of the GPP which should be checked later with Chetto's method.

For each set $\gamma_i$, $d_{i2}$, the deadline of $\tau_{i2}$, is the latest moment at which $\tau_{i2}$ can finish its execution and still leave enough time for $\tau_{i3}$ to be completely executed before its own deadline, $d_{i3}=D_i$, if it is assumed that no other task interferes with it. $d_{i2}$ is therefore

$$d_{i2} = d_{i3} - C_{i3} \tag{1}$$

Since $\tau_{i3}$ may be preempted, this equation merely states the latest deadline for the task in the SPP or, what is equivalent, the latest possible release time for its successor in the GPP. How to ascertain the schedulability of the GPP is explained later.

In what follows, $U$ and $L$ denote the utilization factor of the set $\Gamma_i$ and the set of instants where the schedulability should be checked, respectively.

**Theorem**: In [17] it is proved that set of non-preemptable tasks is feasible under EDF if:

$$U \leq 1, \forall t \in L \tag{2}$$

$$t \geq F(t) = \sum_{i=1}^{n} \max\left(0,1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i + C_p - 1$$

Where $C_p = \max_i\{C_i\}$ and

$$L = \left( \bigcup_{i=1}^{n} \{D_i + kT_i, k \in \aleph\} \bigcap \left[0, \max\left(\max_i\{D_i\}, M\right)\right] \right)$$

The theorem states that for a non-preemptive EDF scheduling to be feasible, the utilization factor should be less than one and the demand in a busy period begining after the task with longer execution time has started its execution should be less than the available time.

*Example*: Let **S**(3)={(2,4,8),(3,6,10),(3,10,10)}. The system is not feasible since:

- $U$=0.85
- $L$={4,6,10,12,16,20,26,28,30,36}
- F(4)=4, F(6)=7 and one task will miss a deadline as can be seen in Figure 1.

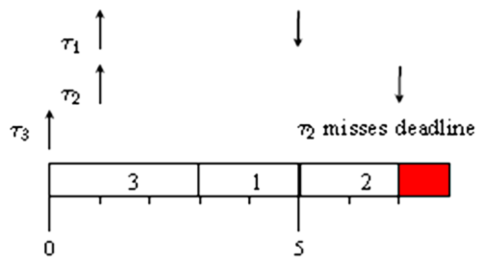

Figure 1. SPP schedulability example

## VI. GPP's Schedulability

The tasks in the GPP are not independent. As explained in the system model, the execution of some of them is conditioned to the completion of others. Traditional approaches to analyse the schedulability are not valid in this case, so the Chetto approach is used [15] instead. In it, the set of precedence related tasks is mapped into a new set of independent tasks with release times and deadlines computed in such a way that the coherence of the set is preserved. In [15] the following is proved:

**Theorem:** Let **S**($m$) be a set of tasks related by precedence. Let $\mathbf{S}^*(m)$ be a set of independent tasks such that they respect the partial order imposed in **S**($m$) for the release and deadline, and $C_i^*=C_i$. **S**($m$) is schedulable if and only if $\mathbf{S}^*(m)$ is schedulable.

**Corollary:** Let **S**($m$) be a set of precedence related tasks, and $\mathbf{S}^*(m)$ the mapped set that respects the partial order imposed by **S**($m$) , then $\mathbf{S}^*(m)$ is schedulable if and only if

$\forall h = 1, \dots, q, \forall g = 1, \dots, q$ such that

$$r_h \leq r_g, d_h \leq d_g \qquad (3)$$

$$\sum_{r_h \leq r_g, d_h \leq d_g} C_s \leq d_g - r_h$$

For simplicity, suppose that the set of precedence related tasks, $\gamma_i$, are composed of just three tasks, with the first and last ones ($\tau_{i1}$, $\tau_{i3}$) executing in the GPP and $\tau_{i2}$ in the SPP. In order to use the above expression it is necessary to determine the release times and deadlines of all the tasks in the chain. The pairs ($r_{i3}$, $d_{i3}$) are already known. Since $r_{i1}=r_i$ is also known, it remains only $d_{i1}$ to be calculated. Some tasks in the GPP are predecessors of tasks in the SPP's and therefore their deadlines should be set in such a way that the successor can have enough time to be executed in the SPP. In order to do this, it is important to know the worst case response time of the tasks in the SPP's.

The response time of $\tau_{i2}$, denoted $R_{i2}$, defined as the maximum time that can elapse between $r_{i2}$ and the end of the execution of $\tau_{i2}$, has to be obtained. The following lemma, proved in [18], shows how to find it:

**Lemma**: The worst case response time of a task $\tau_{i2}$ is found in a deadline busy period in which: a) $\tau_{i2}$ has an instance released at time $r_{i2}$, possibly with others released before. b) all tasks with relative deadline smaller than or equal to $r_{i2}+ d_{i2}$ are released from time $t$=0 on at their maximum rate. c) a further task with relative deadline greater than $r_{i2}+ d_{i2}$, if any, has an instance released at time $t$=-1.

The lemma states that a task in the SPP not only may be blocked by a lower priority task (greater absolute deadline) but also would have to wait for the execution of all the higher priority tasks' instances that are released together and along the busy period. In order to compute the response time, a recursive formula is used until a solution is found:

$$W_i(r_{i2}, t) = \sum_{j \neq i} \min\left\{1 + \left\lfloor \frac{t}{T_j} \right\rfloor, 1 + \left\lfloor \frac{r_{i2} + d_{i2} - d_{j2}}{T_j} \right\rfloor \right\} C_{j2}$$

$$d_{i2} \leq r_{i2} + d_{i2}$$

The above expression computes the demand of the tasks in the interval considered.

$$A_i(r_{i2}) = \max_{d_{i2} > r_{i2} + d_{i2}} \{C_{j2} - 1\} + W_i(A_i(r_{i2}, A_i(r_{i2})) + \left\lfloor \frac{r_{i2}}{T_i} \right\rfloor$$

Computes the length of the actual busy period that has started at $t$=0 previous to the release of $\tau_{i2}$ which is upper bounded by $\Gamma_i$:

$$\Gamma_i = \max_{j: d_{j2} > d_{i2}} \{C_{j2} - 1\} + \sum_{j: d_{j2} \leq d_{i2}} \left\lceil \frac{\Gamma_i}{T_j} \right\rceil C_{j2}$$

The response time is taken as the max between the previous combinations:

$$R_{i2} = \max\{\max\{C_{i2}, A_i(r_{i2})\} + C_{i2} - r_{i2}\}: 0 \leq r_{i2} < \Gamma_i$$

The latest possible deadline for $\tau_{i2}$ is given by the difference between the actual deadline and the response time of $\tau_{i2}$. With $d_{i1}$ set and $R_{i2}$ computed, the latest possible deadline for $\tau_{i1}$ is given by:

$$d_{i1} = d_{i2} - R_{i2} \qquad (4)$$

Since $R_{i2}$, computed by the previous method considers the worst case, it is assured that even if $\tau_{i1}$ finishes as late as its deadline, $\tau_{i2}$ will have enough time to finish before its own deadline.

## VII. Scheduling test

For a system to be schedulable, the GPP and all the SPP's have to be schedulable. Because tasks are not independent it is necessary to determine the release times and deadlines of each one of them and check the schedulability of the processors. The following steps conform the scheduling test:

1. Compute deadlines of the different subtasks according to:
   a. The last task has the deadline of the chain.
   b. If the task's successor is allocated to an SPP, expression (4) is used to determine the deadline. In the case that there is more than one successor, the minimum value is chosen.
   c. If the task's successor is allocated to the GPP, expression (1) is used to determine the deadline.
2. Release times of the different tasks are computed as:
   a. The first task has the release time of the chain
   b. If the task j has a predecessor g: $r_{ij}=r_{ig}+C_{ig}$. In the case that the task has more than one predecessor, the maximum value is chosen.
3. To test the schedulability of the system, each processor has to pass the scheduling test. For the SPP, condition (2) must hold; for the GPP, condition (3) must hold.

*Example*: Let the system be the one presented in Figure 2 and in Table 1. In the figure, each node represents a task, and directed arcs connect predecessors to successors. The system has one GPP and two SPP's. Each chain is already divided in three tasks and the release times and deadlines are those assigned when the system is operating at full frequency on both processors. The processors are coherent and the GPP is schedulable. Table 2 shows the release and deadlines for each subtask as computed by the method. In Figure 3, the evolution of the system during the firsts slots is shown.

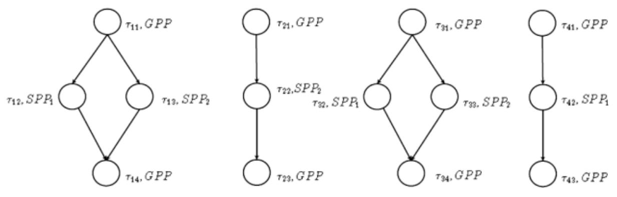

Figure 2. System's tasks graph representation

TABLE 1
SYSTEM'S DEFINITION

| $i$ | $C_{i1}$ | $C_{i2}$ | $C_{i3}$ | $C_{i4}$ | $T_i$ |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 3 | 20 |
| 2 | 1 | 2 | 1 | - | 15 |
| 3 | 1 | 1 | 2 | 2 | 15 |
| 4 | 1 | 2 | 1 | - | 10 |

TABLE 2
COMPUTED DEADLINES AND RELEASE TIMES

| $i$ | $r_{i1}$ | $r_{i2}$ | $r_{i3}$ | $r_{14}$ | $d_{i1}$ | $d_{i2}$ | $d_{i3}$ | $d_{14}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 2 | 7 | 12 | 17 | 17 | 20 |
| 2 | 1 | 2 | 6 | - | 10 | 14 | 15 | - |
| 3 | 1 | 2 | 2 | 5 | 11 | 13 | 13 | 15 |
| 4 | 1 | 2 | 4 | - | 7 | 9 | 10 | - |

## VIII. Energy consumption management

In CMOS technology, extensively used in today processors, the energy consumed in the execution of a given task diminishes quadratically with voltage [19]. Assuming that there is an available continuous spectrum of frequencies, the minimum frequency at which each processor is preemptible EDF-schedulable, is given by: $f_{oj}=f_{nj}U_j$. $f_{oj}$ and $f_{nj}$ denote the operating and the nominal frequencies of processor $j$, respectively and $U_j$ the utilization factor of the processor defined as $\sum C_i/T_i$. By reducing the operating frequency, the execution time of the tasks is incremented and a new feasibility test should be performed for the case of the non-preemptible processors to check the schedulablity.

In real microprocessors, however, the operating frequencies cannot be varied continuously. Each processor shall use, therefore, an operating frequency chosen from the available set in such a way that it is equal to or bigger than the one obtained from the continuous spectrum. In that way, a set of suboptimal single frequencies produced by Static Voltage Scaling is obtained. Two important problems associated with Dynamic Voltage Scaling in real-time systems are avoided: the appearance of scheduling anomalies leading to not meeting time-constraints and the need to introduce transit times between frequencies as an overhead in the calculations when scheduling with multiple frequencies.

If $|f|$ and $p$ denote the number of available frequencies in the processors and the number of processors, respectively, the number of possible frequency assignments is $|f|^p$ and the problem may become intractable.

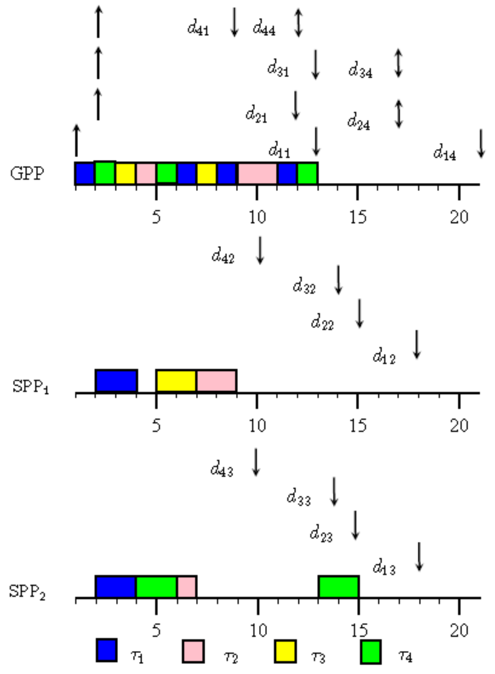

Figure 3.Temporal Evolution, ↑release, ↓ deadline

However, since in practice $|f|$ and $p$ are small numbers, a systematic search of feasible frequency combinations may be conducted and heuristic or statistical methods result unnecessary. In fact, many possible solutions are not considered because the utilization factor of the processors at nominal frequency prevents them. The search is static off-line, it is made only once and requires a small time. For example, the calculations for a set of thirty chains processed in a five-core SoC, performed in a Celeron 2.4 GHz PC, require only 60ms.

## IX. Energy consumption computation

In what follows it is assumed that a satisfactory assignment of tasks over the GPP and the SPPs has been made and that the system, with all its processors operating at nominal frequency, meets all the time-constraints.

**Theorem:** The energy consumed per unit of time in a multiprocessor system operating at reduced frequencies is given by:

$$E_o=\sum_{j=1}^{p} f_{oj}^2\left(U_j+\alpha(1-U_j)\right) \qquad (5)$$

where $f_{oj}$ is the reduced frequency of processor $j$, $U_j$ is the utilization factor of processor $j$ at nominal frequency, and α is a constant less than or equal to 1 that represents the percentage of energy consumed by the processor when it is idle.

**Proof**: To compute the energy consumption per unit of time it is necessary to determine the amount of time the system is either busy or idle. To do that, the concepts of work-function and utilization factor are used. The work-function at instant *t* is defined as:

$$\omega(t) = \sum_{h=1}^{h=m} \left\lceil \frac{t}{T_h} \right\rceil C_h$$

If *M* denotes the value of the Least Common Multiple of the periods, often called the hyperperiod, $\omega(M)$ and $(1 - \omega(M))$ represent the time the processor spends executing tasks or is idle in the hyperperiod, respectively. If the power of the processor at nominal frequency is taken as unit of power, the energy consumed in the hyperperiod will therefore be:

$$E = \omega(M) + \alpha(1 - \omega(M))$$

The expression:

$$\frac{\omega(M) + \alpha(1 - \omega(M))}{M} = U_p + \alpha(1 - U_p)$$

represents then the average energy consumed per unit of time at nominal frequency. Knowing the power of the processor in W and the unit of time used, it is a trivial matter to obtain the energy consumed in MKS units (e.g. Ws). Since the power consumption undergoes a cubic reduction with frequency but the time necessary to execute a given workload is inversely proportional to the frequency, the average energy consumption when the processor operates at $f_o$, relative to the nominal consumption is

$$\left(\frac{f_o}{f_n}\right)^3 \frac{U_p}{(f_o/f_n)} + \frac{1 - U_p}{(f_o/f_n)} = \left(\frac{f_o}{f_n}\right)^2 \left(U_p + \alpha(1 - U_p)\right)$$

If $f_n$ is taken as unit of frequency, the final expression is:

$$f_o^2\left(U_p + \alpha(1 - U_p)\right)$$

The energy consumed by all processors in the chip will be the sum of the individual consumptions.

In this paper, the energy consumed in the idel state is assumed to be 15% ($\alpha$=0.15) of that consumed when executing tasks, a figure based on measurements presented at [20]. If $E_n$ denotes the consumption with all the processors operating at nominal frequency, the relative saving is defined as $(E_n-E_o)/E_n$.

## X. Frequency assignment method

Unfortunately the method to determine operating frequencies in the case of systems of precedence-related tasks executed in different processors is not so straight: diminishing the frequency in the processor in which the predecessor-task is executed may result in a postponement of its deadline leading to an inadmissible delay in the release time of the successor. The problem can be represented in a p-dimensional space in which each dimension is associated to the frequency of one processor. Each possible solution in this space can be represented as an *n*-tuple where each element represents the actual operating frequency of each processor. In this view, the *p*-tuple that minimizes the energy consumption has to be found. The procedure is iterative and it is described in the following algorithm:

1. Eliminate the *p*-tuples with frequencies below the utilization factors of each processor as they are not possible solutions.
2. Compute $E_o$ for each one of the remaining *p*-tuples.
3. Order the *p*-tuples by increasing values of $E_o$.
4. Select the *p*-tuple with the minimum $E_o$.
5. Check the schedulability of the system following the algorithm of the previous section. If the system is schedulable a solution has been found.
6. If the system is not schedulable, select the next *n*-tuple in the ordered list and return to step 5.

*Example*: Consider the system described in the example of the previous section. The total utilization factor of the GPP is 0.73 and that of both SPPs is 0.37. The available frequencies are $f_n$, $0.75f_n$ and $0.5f_n$.The minimum available frequencies for the GPP and the SPP computed independently are equal to $0.75f_n$ and $0.5f_n$. However, if the SPP frequencies are set to that value, the GPP will not be schedulable. By incrementing the SPPs' frequencies to $0.75f_n$ the system becomes feasible. The system will be then described by Tables 3 and 4.

TABLE 3
COMPUTATION TIMES RECOMPUTED FOR THE REDUCED FREQUENCIES

| $i$ | $C_{i1}$ | $C_{i2}$ | $C_{i3}$ | $C_{i4}$ |
|---|---|---|---|---|
| 1 | 1.33 | 2.66 | 2.66 | 4 |
| 2 | 1.33 | 2.66 | 1.33 | - |
| 3 | 1.33 | 1.33 | 2.66 | 2.66 |
| 4 | 1.33 | 2.66 | 1.33 | - |

TABLE 4
RELEASED AND DEADLINES MODIFIED

| $i$ | $r_{i1}$ | $r_{i2}$ | $r_{i3}$ | $r_{14}$ | $d_{i1}$ | $d_{i2}$ | $d_{i3}$ | $d_{14}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2.33 | 2.33 | 7.33 | 9 | 16 | 16 | 20 |
| 2 | 1 | 2.33 | 8.33 | - | 7.33 | 13.66 | 15 | - |
| 3 | 1 | 2.33 | 2.33 | 7.33 | 7.66 | 12.66 | 12.66 | 15 |
| 4 | 1 | 2.33 | 4 | - | 7 | 8.66 | 10 | - |

Using expression (5) to compute the energy consumed at nominal and reduced frequencies, an overall energy saving of 43% is obtained.

## XI. Esperimental Evaluation

The performance of the method presented in the previous section was evaluated by simulations carried out on a system composed of one GPP and four SPP's. The set of available frequencies, relative to the nominal one, in each processor was {1.00, 0.75, 0.50, 0.25}. Workloads were randomly generated and the energy consumption to process them at nominal and at reduced frequencies were determined. The utilization factors of the SPP's varied between 0.20 and 0.7 in steps of 0.1. For every case, 100 basic workloads were run. A basic workload is a set of 30 chains with periods randomly chosen in the interval [3000, 30 000] with uniform distribution. The second tasks in the chain were allocated to the SPP's in such a way that balanced loads were obtained among them. Loads were run and the energy savings determined.

The results are shown in Figures 4- 9. SPP's utilization factors are fixed at different values (0.2 to 0.7) while the GPP's utilization factor is varied between 0.1 and 0.9.

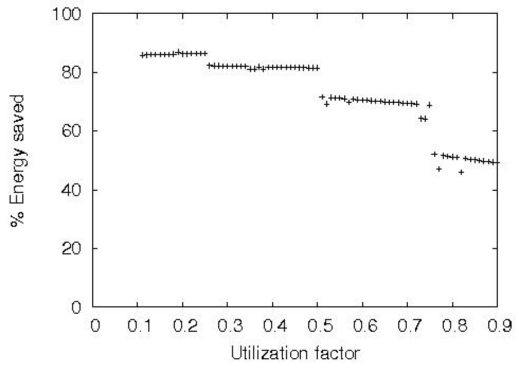

Figure 4. $U_{SPP}$=0.2

The experiments show that the method provides relative energy savings that depend on the utilization factors of the processors. The perceptual saving is about 90% for low utilization factors ($U_{GPP}$=0.1; $U_{SPP}$=0.2) and falls to near 20% for high utilization factors ($U_{GPP}$=0.9; $U_{SPP}$=0.7). This is due to the fact that when the operating frequency, although less than 1, is bigger than 0.75, the GPP must operate at nominal frequency with no relative energy saving. However, even when the SPPs have a high average utilization factor (0.7, Figure 6), they can probably work at 0.75 producing the little savings shown.

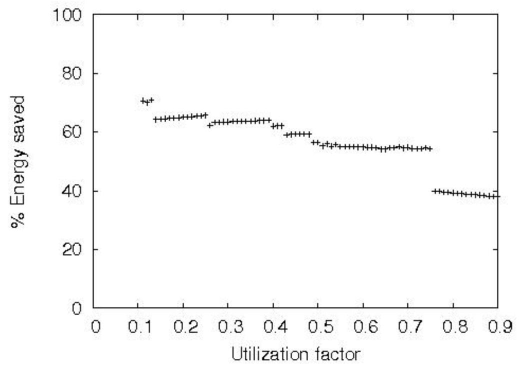

Figure 5. $U_{SPP}$ =0.3

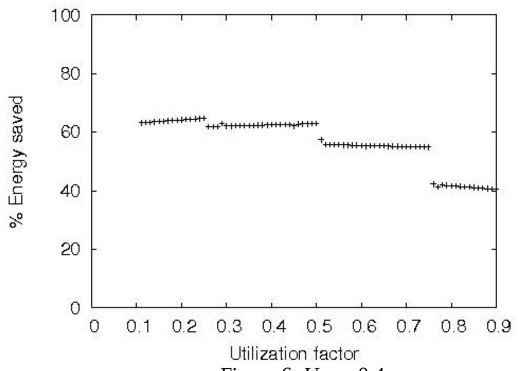

Figure 6. $U_{SPP}$ =0.4

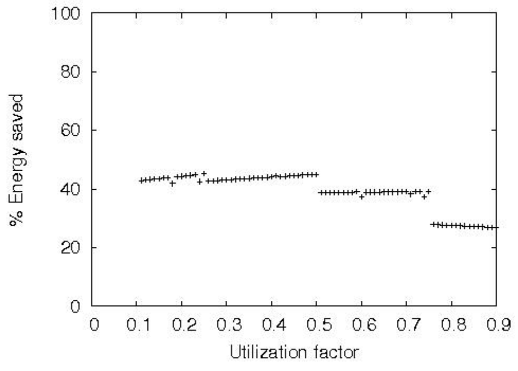

Figure 7. $U_{SPP}$ =0.5

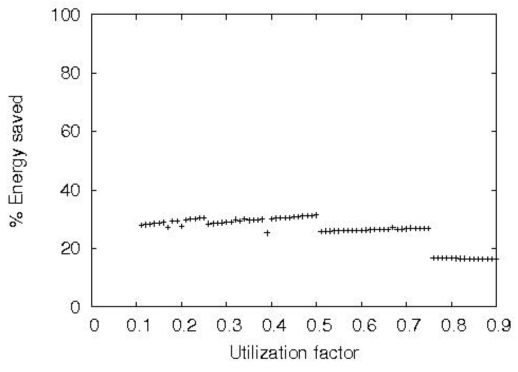

Figure 8. $U_{SPP}$ =0.6

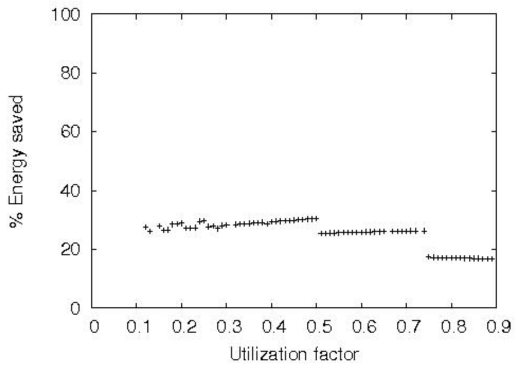

Figure 9. $U_{SPP}$ =0.7

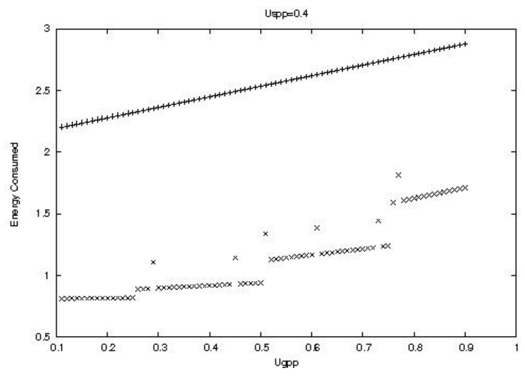

Figure 10. Energy Consumption with nominal and reduced frequencies
o Nominal Frequency, + Reduced

Four regions, approximate plateaus with low gradients, corresponding to the four available frequencies at the GPP, can be observed in every case. As could be expected, abrupt incremental steps take place when changing those frequencies. The plateaus representing the relative energy savings ($E_n$-$E_o$/$E_n$) have very small derivatives, indicating slight increments or decrements in the relative savings as the GPP's utilization factor increases.

The reason behind this result can be understood by examining Figure 10. In it, actual energy consumptions are represented *vs* the GPP utilization factor. The utilization factors of the SPP's are kept constant at a value of 0.4 throughout the experiment. As can be seen, the energy consumed by the system when all processors operate at nominal frequencies, $E_n$=$f(U_g)$, is a linear function of the GPP utilization factor. This could be expected since, bearing in mind expression (5), the consumption at nominal frequencies ($f_{oj}$ = $f_n$ = 1), can be expressed as:

$$E_n = U_g + 0.15(1 - U_g) + K_s = 0.85U_g + K_s \qquad (6)$$

where $K\{s\}$ denotes the consumption of the SPP's, constant throughout the experiment. The energy consumed at reduced operating frequencies, $E_o$=$f(U_g)$, is shown in the four discontinuous steps. For low utilization factors (0.10 to 0.25), the GPP can operate at a frequency of 0.25. From factors of approximately 0.25 up, the system is not feasible any more, and the GPP must jump to an operating frequency of 0.5, producing the discontinuity and the second step. Further discontinuities are produced for utilization factors of approximately 0.50 and 0.75, and the steps corresponding to the GPP operating at higher available frequencies are produced. At every step the consumption grows with a higher derivative as the utilization factor grows and less slack is available.

However, the relative savings shown in Figures 4-9 are the representation of 1-($E_o$/$E_n$) *vs.* the GPP utilization factor. This means that, in spite of growing energy consumptions, relative savings may grow, be constant or diminish according to $E_o$/$E_n$ being a decreasing, constant or increasing function; in other words, according to $d(E_o/E_n)/dU_g$ being negative, constant or positive, respectively. The function $E_o$/$E_n$ depends on the type of load, the utilization factors and the available frequencies.

## XII. Conclusions and future work

In this paper a method to deal with the schedulability of precedence-related tasks in multicore GPP/SPP's systems-on-a-chip has been presented. The method is completed with an heuristic based frequency selection for the processors involved to reduce the energy consumption.

The system model follows the industry trend: one General Purpose Processor acting as a master of one or many Special Purpose Processors acting, in turn, as accelerators of the first one. Tasks executing in the chain GPP-SPP-GPP are precedence-related. The scheduling problem is compounded by the fact that the specialized processors act as critical sections and need, therefore, a non-preemptive scheduling discipline.

Scaling algorithm is used to avoid shortcomings like the the need to introduce transition times between frequencies in the schedulability calculations. Extensive simulations were performed. They showed that important savings in energy consumption may be obtained. Future work will be oriented to the determination of heuristics to find a suboptimal set of frequencies in systems of such complexity that a systematic search of all possible sets is not feasible.

## References

[1] J. M. Paul, What's in a name?, *IEEE Computer* 39 (3) (2006) 87–89.

[2] *Texas Instruments, Military Semiconductor Products Fact Sheet* SMJ320C80 (2000).

[3] C. Gonsalves, *Challenges of providing interoperable solutions in todays IPTV market*, Texas Instruments (April 2006).

[4] Atmel, *Atmel Smart Internet Appliance Processor*, http://www.aplio.com/pdf/AplioTRIO Summary DS Rev01.pdf.

[5] S. K. Moore, Winner multimedia monster, *IEEE Spectrum*.

[6] J. H. Anderson, S. K. Baruah, *Energy-aware implementation of hard-real-time systems upon multiprocessor platforms*, in: Proc. ISCA 16th International Conference on Parallel and Distributed Computing Systems, 2003, pp. 430–435.

[7] S. Rivoire, M. Shah, P. Ranganathan, C. Kozyrakis, J. Meza, Models and metrics to enable energy efficiency optimizations, *Computer* 40 (12) (2007) 39–48.

[8] R. Rajkumar, Ch. Synchronization in multiple processors systems *Synchronization in Real-Time Systems: Apriority Inheritance Approach*, Kluwer Academic Publishers, 1991,.

[9] P. Gai, L. Abeni, G. Buttazzo, *Multiprocessor dsp scheduling in system-on-a-chip architecture*, in*:* 14th Euromicro Conference on Real-Time Systems, 2002.

[10] I.-G. Kim, S.-K. Choi, K-H.and Park, D.-Y. Kim, M.-P. Hong, *Real-time scheduling of tasks that contain the external blocking intervals*, in: 2nd International Workshop on Real-Time Computing Systems and Applications, 1995, pp. 25 – 27.

[11] N. Audsley, A. Burns, M. Richardson, K. Tindell, A. J. Wellings, Applying new scheduling theory to static priority pre-emptive scheduling, *Software Engineering Journal*.

[12] R. Jejurikar, *Energy aware non-preemptive scheduling for hard real-time systems*, in: Proc. 17th Euromicro Conference on Real-Time Systems, IEEE Computer Society Press, 2005, pp. 21–30.

[13] W. Li, K. Kavi, R. Akl, A non-preemptive scheduling algorithm for soft real-time systems, *Computers & Electrical Engineering* 33 (1) (2007) 12–29.
[14] G. L. Liu, J. W. Layland, Scheduling Algorithms for Multiprogramming in Hard Real Time Environment, *ACM* 20 (1973) 46–61.
[15] H. Chetto, M. Silly-Chetto, T. Bouchentouf, Dynamic scheduling of real-time tasks under precedence constraints, *Real-Time Systems* 2 (1990) 181–194.
[16] K. Jeffay, D. F. Stanat, C. U. Martel, *On non-preemptive scheduling of periodic and sporadic tasks*, in: Proceedings of the IEEE Real-Time Systems Symposium, 1991, pp. 129–139.
[17] Q. Zheng, K. G. Shin, On the ability of establishing real-time channels in point-to point packet-switched networks, *IEEE Trans. on Communications* 42 (2/3/4) (1994) 1096–1105.
[18] L. George, N. Rivierre, M. Spuri, Preemptive and non-preemptive real-time uniprocessor scheduling, Tech. Rep. 2966, *INRIA* (September 2006).
[19] P. Pillai, K. G. Shin, *Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems*, in: Proc 18th Symposium on Operating Systems Principles, 2001, pp. 89–102.
[20] J. Pouwelse, P. Langendoen, H. Sips, *Dynamic voltages scaling on a low-power microprocessor*, in: Proc. 7th Conf. on Mobile and Computing and networking MOBICOM01, 2001.

## XIII. Authors' information

**Rodrigo Santos** received his Engineering degree in 1997 from Universidad Nacional del Sur and got his Ph.D., degree in Engineering in 2001. He has become a Researcher for The Argentina in 2005 and in the same year became Assistant Professor His research interests are mainly related to real-time systems: QoS, Multimedia, Operating Systems and Communications. He has published his research's results in international indexed Journals and proceedings of Conferences. He is a member of several Technical Committees for conferences in the area of real-time systems and also a reviewer for several journals. He is the President for the Latin American Center of Studies in Informatics and a member of the Working Group 6.9 of IFIP. He is also an IEEE member.

**Jorge Santos** is Consulting Professor at the Department of Electrical Engineering and Computers, Universidad Nacional del Sur, Argentina, where he teaches courses on Real-Time Systems and on Professional Communication. He has published more than 70 papers on multivalued logics and their electronic implementation, theory of automata, systems reconfigurations, local area networks, and real-time systems. He has been head of the department and principal researcher of the National Council of Scientific and Technical Research. He is Corresponding Member of the Argentine Academy of Engineering.

**Javier Orozco** received his Electrical Engineering degree in 1984 and his doctoral degree in Engineering in 1998 from National Universidad Nacional del Sur, Bahia Blanca, Argentina. He joined the Argentine National Research Council in 1991as a researcher. Currently is a Professor at the Department of Electrical Engineering and Computers at Universidad Nacional del Sur. His main interest area is in Real-Time operating systems theory and applications and architectures for embedded systems and communications. He has an active participation in several Scientific and Technical Committees and governmental boards for science and engineering promotion. He has published on local area networks, digital architectures and real-time systems theory and applications.

**David Donari** received his Computer Engineering degree in 2006 from Universidad Nacional del Sur. Presently he has a Doctoral Scholarship from the Argentine National Research Council. His main interest areas are in Real Time Operating Systems and Embedded Systems.

**Leo Ordinez** received his Computer Engineering degree in 2006 from Universidad Nacional del Sur. Presently he has a Doctoral Scholarship from the Argentine National Research Council. His main interest areas are in Real Time Operating Systems and Embedded Systems.