smartenum: A Branch-and-Bound Algorithm for Optimum Frequency Set Establishment in Real-Time DVFS

E. B. Valentin[1], **R. S. Barreto**[1]

[1]Department of Computer Science
Federal University of Amazonas
Manaus-AM, Brazil

{ebv,rbarreto}@dcc.ufam.edu.br

Abstract. *This paper describes an offline branch-and-bound algorithm to establish the optimum frequency set to execute real-time tasks taking into account worst-case scenarios on the DVFS technique. The real-time tasks are scheduled by a dynamic fixed priority scheduler, such as Rate-Monotonic. The task model also considers mutual exclusion relations adopting the Priority Ceiling Protocol. The schedulability analysis is carried out by response time technique, which had to be adjusted to consider several frequencies. Two case studies are detailed. Results have shown a reduction of* 91% *and* 78% *in the number of evaluated configurations. In addition, experimental results pointed out energy reductions of* 38.79%*, and* 30.46%*.*

1. Introduction

Modern processors design provides the possibility to change operating frequency dynamically at runtime. Therefore, clock speed and corresponding voltage may be dynamically controlled to the lowest available level while meeting task's timing constraints. This is the key idea behind a technique known as *Dynamic Voltage and Frequency Scaling* - DVFS. When considering processor circuitry, there is a quadratic relationship between energy consumption and voltage. This relation is described on Equation (1):

$$E = C_l \times N_{cycle} \times V_{dd}^2 \tag{1}$$

where E is energy, C_l is circuitry capacitance, N_{cycle} is number of cycles and V_{dd} is the voltage [Gutnik 1996].

In some situations, you can lower the supply voltage in such a way to take advantage from this quadratic relation between voltage and energy consumption. However, lowering the supply voltage also reduces linearly the clock speed. For instance, consider a task with 25ms deadline executing in a 50MHz processor under 5V. If this task requires 5×10^5 execution cycles, the processor finishes its execution within 10ms and stays idle for the remaining 15ms. However, if user scales processor speed and voltage down to 20MHz and 2V, the processor finishes this task exactly at its deadline, 25ms, resulting in a 84% energy consumption reduction [Shin et al. 2001].

Although this technique can yield meaningful energy consumption reduction, its usage requires care, especially when timing constraints are considered. This problem may be trivial if each task is taken into account isolated without any concern of other tasks interference. Once it is known the task's deadline and the required worst-case execution cycles, the best execution frequency can be selected in order to achieve lowest energy consumption and still reach the deadline. On the other hand, when a system with more than one task is considered, this problem becomes more complicated.

An interesting problem arises when executing such systems in DVFS-enabled processors: *"In which frequency each task must be executed, so that the whole system reaches minimum energy consumption from processor circuitry and all tasks meet their deadlines?"* In this case, for each frequency combination, a new schedulability analysis is required, because a single change into one task execution time will reflect the whole system, because interference between every tasks will suffer modification. If the system has N real-time tasks and the processor has Γ possible frequencies, then there are Γ^N possible frequency combinations.

The proposed method is an offline branch-and-bound algorithm to establish the optimum frequency set to execute real-time tasks taking into account worst-case scenarios on the DVFS technique. This is the main contribution of this paper. So far, the algorithm only considers pruning unschedulable nodes. Real-time tasks are scheduled by a dynamic fixed priority scheduler, such as Rate-Monotonic or Deadline-Monotonic. The task model also considers mutual exclusion relations adopting the priority ceiling protocol (PCP). Thus, each task can be interfered by factors like high priority tasks and shared resources being held by low priority tasks. A schedulability analysis is performed in order to check if these interferences can lead the system to an undesirable state in which a real-time task would miss its deadline.

This work is part of a larger project that aims to propose a framework to help designers to develop embedded multimedia applications with low energy consumption for wireless portable devices. Although this work considers worst-case execution scenarios, it is worth mentioning that this is only the starting point since there is ongoing projects that takes into account the actual execution scenarios that exploits slack time.

This text is structured as follows: related works are reviewed in Section 2, the problem is modeled in Section 3, the proposed algorithm is presented in Section 4, experimental results are then analyzed in Section 6 and conclusions and future works are discussed in Section 7.

2. Related Works

Problems related to energy consumption have been solved by using several known techniques. Havinga [Havinga 1997] has produced a survey about most of them. Dynamic voltage and frequency scaling, idle and sleep operating modes, dynamic power management (DPM), clock regions, co-processors for specific application type, and operating system tuning are some of discussed techniques. It is worth mentioning that

while implementing those techniques, usually, application's timing constraints are not taken into account.

Zhao and Aydin [Zhao and Aydin 2009] proposed a combined approach between DVFS and DPM techniques to reduce energy consumption on real-time systems. Although a worthy work, they propose deal with only one task leading to a non-preemptive solution. An important contribution in the power-aware scheduling of real-time tasks was done by [Mejia-alvarez et al. 2004]. They propose a solution based on knapsack problem considering system utilization factor. Nevine in [AbouGhazaleh et al. 2003] proposes a colaborative solution which involves scheduler and compiler utilizing intra-task DVFS. However, both works do not take shared resource into account. Choi in [Choi and Pedram 2005] presents an intra-process approach to making use of runtime information about the external memory access statistics in order to perform CPU voltage and frequency scaling with the goal of minimizing the energy consumption. But in this case, it does not consider real-time tasks. An inter-task DVFS approach is done by Yao in [Yao et al. 1995]. This work is based on time slices. It is proposed a mechanism to determine the optimum execution frequency for N tasks in each time slice. It is also proposed a modification on the Earliest Deadline First (EDF) algorithm in order to cover the proposed method. Another strategy to explore DVFS usage is to insert frequency scaling instructions in specific points of tasks code. Azevedo in [Azevedo et al. 2002] proposes an intra-task DVFS based to determine these points and which frequencies are required. The points are chosen based on code profiling and simulations. Its simulation results have shown an energy reduction of about 60%. However, the code profiling is hard to produce and usually application dependent. Shin in [Shin et al. 2001] presents an intra-task DVFS solution based on static analysis. Task code is evaluated and analyzed in order to produce a control flow graph. Each node represents a basic block and contains information about the number of work case execution cycles. This graph is then utilized to determine which points in the code are eligible to change frequency, based on the number of not executed cycles. That solution does not consider preemption in the system. Pillai and Shin [Pillai and Shin 2001] present a class of novel algorithms called real-time DVS (RT-DVS) that modify the OS's real-time scheduler and task management service to provide significant energy savings while maintaining real-time deadline guarantees. Shared resources are not considered in that work.

3. Problem Definition and Modeling

Consider a processor $\mathcal{P}$ with DVFS feature available for Γ frequencies. The set of all possible frequencies in $\mathcal{P}$ is $F = \{f_i \mid f_i \text{ is an available frequency in Hz for } \mathcal{P} \text{ and } 1 \leq i \leq \Gamma\}$. Consider also a model $\mathcal{M}$ with N tasks. In $\mathcal{M}$, the system is executed under a dynamic fixed priority scheduling.

Consider now the set β consisting of N-tuples. Let $K \in \beta$, $K = \langle k_1, k_2, \ldots, k_N \rangle$, where $k_i \in F$. K represents a possible configuration of frequencies assignment to T_i tasks in $\mathcal{M}$, where T_i is executed with frequency k_i. Each ele-

ment $K \in \beta$ must be interpreted as an arrangement of the possible assignments that can be created using all available frequencies in $\mathcal{P}$. Thus, elements in β are possible configurations for execution of $\mathcal{M}$. There are Γ^N possible configurations K's in β.

Due to shared resource there is the need to consider mutual exclusion relations in tasks of $\mathcal{M}$. $T_i \in \mathcal{M}$ has the following properties: D_i is its deadline; P_i is its period of execution; $WCEC_i$ is its worst-case execution cycles; $C_i(f)$ is T_i's execution time which is function of frequency f. $C_i(f) = \frac{WCEC_i}{f}$; p_i is T_i's priority; J_i is the *release jitter* for task T_i, which indicates the worst release case for T_i; $I_i(K)$ is the interference suffered by T_i, which is function of frequency set K. $I_i(K)$ corresponds a time window in which there is continuous execution of tasks with greater or equal priorities to T_i's; $R_i(K)$ is T_i's response time as function of frequency set K. $R_i(K) = I_i(K) + J_i$. $I_i(K)$ is result of a schedulability test. The schedulability test is based on response time. It can be calculated using Equation (2).

$$I_i^{n+1}(K) = C_i(k_i) + B_i + \sum_{j \in hp(i)} \left\{ \left\lceil \frac{I_i^n(K) + J_j}{P_j} \right\rceil \times [C_j(k_j) + \sigma] \right\} \tag{2}$$

In Eq.(2), $hp(i)$ is the set of tasks which has higher priorities than T_i. $\frac{I_i^n(K)+J_j}{P_j}$ represents the number of occurences of T_j over I_i. σ is the overhead caused by the frequency and voltage switch process. B_i represents the time spent in shared resources locks. In order to deal with priority inversion problem, B_i is calculated considering the Priority Ceiling Protocol (PCP) [Sha et al. 1990]. Sha *et al.* [Sha et al. 1990] does not treat with different available frequencies. It is worth noting that Eq.(2) has been rewritten considering the selected frequency for execution of each task T_i. From this definition, it is noticeable the relevance of properly selecting a frequency to a task. Because selecting an ordinary frequency f to a specific task T_i, it will distinguish T_i's execution time $C_i(k_i)$ and consequently its response time $R_i(K)$. In this sense, a task is always completelly defined as function of its selected frequency. Another property derived from the frequency choice is the task idle time. T_i's idle time is $D_i - C_i(K)$. When the target is to maximize processor utilization time, the selected frequency is considered optimum when T_i's idle time is the minimum. However, there is an intrinsic condition on this problem, because all tasks must meet their timing constraints, in this case D_i. Hence,

$$\forall\, i < N,\ R_i(K) \leq D_i \tag{3}$$

Elements in β must be evaluated by means of response time schedulability test. In this case, $K \in \beta$ is considered a feasible configuration if, and only if, it does not violate the condition (3). The problem consists in finding the element K in β which leads the system to a configuration where constraints (3) are satisfied and the lowest possible consumption in $\mathcal{P}$ is achieved.

The objective is then to determine $M \in \beta$ so that:

$$Minimize\{idle(M) = \sum_{i=1}^{N} (D_i - C_i(k_i))\} \quad (4)$$

in such a way that, using M, $R_i(M)$ respects condition (3). $idle(M)$ is referred in this paper as *idle time* of frequency configuration M. Thus, $idle(M)$ represents the maximum utilization of idle times in each task.

4. *Branch-and-Bound* Algorithm to Establish Optimum Frequency Set

The proposed algorithm, here called *smartenum*, generates a search tree for all possible arrangements of tasks in $\mathcal{M}$ and frequencies in F. The idea behind constructing the search tree is to put each task in a specific level of the search tree and, associated with this task, to determine its best frequency. This way, the search tree will be limited up to N levels. In the proposed algorithm the search tree is traversed using a depth first search method. The stop condition is the impossibility of satisfying condition (3). The algorithm performs two initial prunings, throwing away frequencies that alone produces computation time which violates (3) and establishing a starting point for the search. These prunings correspond to a upper local limit and a lower limit for the combinatorial space. Figure 1 shows its diagram.

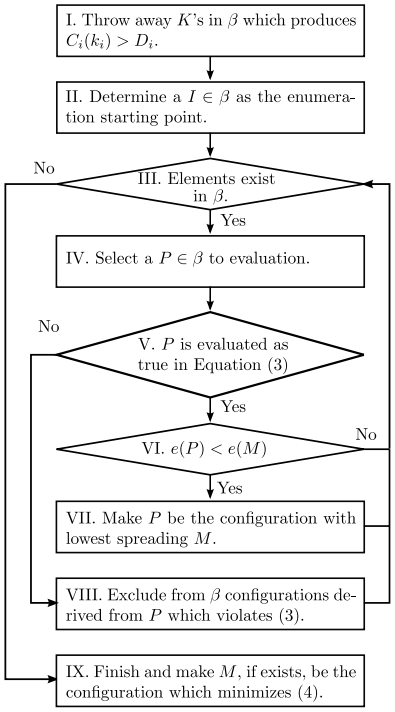


Figure 1. *smartenum* Algorithm Diagram

The proposed algorithm in Figure 1 has two operation types: enumeration and pruning. Enumeration operation consists of listing and evaluating elements $K \in \beta$. The evaluation is the schedulability test in order to check constraint (3) for that configuration K. Pruning operation consists of eliminating elements in order to reduce the number of elements to verify during the enumeration operation.

All available frequencies are evaluated for each task in a decreasing order, so that the greater computation times are considered only as a last resource. This premise allows to throw away lower frequencies, in case an ordinary frequency has been found as useless to one specific task. This evaluation method compounds the search tree illustrated in Figure 2. Thus, all frequencies related to task T_i are at level i of search tree. And every path starting from level 1 and ending at level N represents an element

$K \in \beta$. It is worth emphasizing that, even though this algorithm uses the idea of search tree, it is not required to instantiate all elements in memory. It is possible to generate every element K in an iterative fashion considering all frequencies in each level.

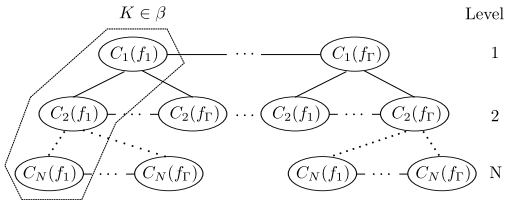


Figure 2. Search tree used by *smartenum*

In Figure 2, $f_1 > f_2 > \ldots > f_\Gamma$ and $C_i(f_1) < C_i(f_2) < \ldots < C_i(f_\Gamma)$. While evaluating frequencies in a decreasing order, it is possible to perform steps (I) and (II) of initial pruning, from Figure 1. The pruning step (I) comprises evaluation of all frequencies in F for each task, individually. While evaluating a single task, it is possible to take into account the local constraint (5).

$$C_i(k_i) \leq D_i \tag{5}$$

If an ordinary K has a k_i which violates the local condition (5), hence K also violates the restriction (3). The pruning in step (I) corresponds to perform this local analysis and to eliminate K's which do not respect the condition (5). This means to establish an upper local limit for each task. This limit determines what is the maximum computation time each task can assume and also the lowest frequency that can be applied on it, taking into account only properties from that task. It is worth mentioning that this pruning does not throw away all K's in β which violate (3). There are K's that meet local restriction (5) for all tasks but fail to satisfy (3) due to the existence of interference between them. This pruning step does not perform any kind of direct evaluation of the whole configuration.

The pruning step (II) corresponds to finding a starting point for the enumeration operation. It aims to determine a lower limit I. This pruning procedure considers the set $\beta' \subseteq \beta$, where $\beta' = \{K_i \mid K_i = \langle k_i, k_i, \ldots, k_i \rangle\}$. β' is a set of configurations where all tasks executes at same frequency. Elements in β' hold the property (6).

$$idle(K) \leq idle(K') \Leftrightarrow k_i \leq k'_i,\ K_i = \langle k_i, k_i, \ldots, k_i \rangle,\ K'_i = \langle k'_i, k'_i, \ldots, k'_i \rangle \tag{6}$$

As the frequency k_i of configuration K is lower than k'_i of configuration K', then computation times $C_i(k_i)$ of all tasks T_i in configuration K will be greater than equivalents computation times $C_i(k'_i)$. As consequence, idle times in configuration K are lower than in configuration K', thus $idle(K) \leq idle(K')$. Taking into account the property from Equation (6), this pruning step performs a binary search in β'. Thus, the aim of this pruning is to find $I \in \beta'$ which does not violate the condition (3) and produces the lowest idle time of all configurations in β'. This pruning step evaluates configurations in order to perform the search. The following algorithm's steps consist of enumeration itself. In the worst case, even if all pruning steps are applied, all elements $K \in \beta$ are evaluated. The stop condition (III) of this enumeration is while there exist elements

$K \in \beta$ not evaluated. Step (IV) selects an element P for evaluation. It traverses the tree present in Figure 2 from left to right and from bottom to up. Frequencies are assigned and fixed for each higher levels, while frequencies are decreasingly variated in lower levels. At beginning of the search, the highest available frequency in each level is fixed as starting point. Next P's elements are determined by decreasingly variating only the frequency on last level of the tree, in this case level N. Once all frequencies on level N are already evaluated, the algorithm returns it to the highest frequency available and on level $N-1$ it selects the next available frequency, in decreasing order. This process is repeated for all N levels of the search tree. When all frequencies on level 1 are evaluated, or all elements $K \in \beta$ have been evaluated, this means that the whole search tree has been transversed. This situation represents the stop condition of this enumerative search.

In the enumeration process, each selected configuration P carries out the response time test. This test is represented by (3), which corresponds to step (V). If a selected configuration P is feasible, then the algorithm checks if it produces the lowest configuration idle time so far. This is performed at step (VI), that is, if $idle(P) < idle(M)$. If yes, P becomes the new configuration M with lowest idle time. This part is in the step (VII).

There is the possibility to apply some prunings when a selected configuration P violates condition (3). This pruning is performed on step (VIII) of the algorithm. This pruning relies on the fact that on all levels the computation time is evaluated in an increasing manner. So, having fixed frequencies at levels 1 to $N-1$, if a frequency f_i on level N is considered to produce a not feasible configuration P, then, a frequency $f_{i+1} < f_i$ will produce another configuration P', also not feasible. Thus, there is no need to evaluate P'. This pruning is performed in a similar way for all levels. This pruning operation adopts the result from evaluation done by the enumeration operation in order to execute prunings. The step (VIII) is the *bound* process on this algorithm. Step (IX) is the end point in this algorithm. If it exists one, this step reports a feasible configuration M that produces the lowest possible idle time.

5. Exemplification

This section details how to apply the proposed algorithm in a small case study. Let a model $\mathcal{M}$ with two tasks executed in a processor with four frequencies available and sharing three resources. The available frequencies and voltages are listed in Table 1(a). The part of the task model with percentage of resource usage are shown in Table 1(b).

Table 1. Case study for exemplification.

(a) Freq. and Voltages

i	Frequency	Voltage
1	$600MHz$	$1.19V$
2	$466MHz$	$1.05V$
3	$333MHz$	$0.91V$
4	$80MHz$	$0.72V$

(b) Task model

				Resource usage		
Task	p_i	Cycles	D_i	i	ii	iii
1	0	1500	30	0%	40%	40%
2	1	900	10	20%	20%	0%

Table 2. Complete enumeration

									Can be pruned by		
f_1	f_2	K#	$C_1(s)$	$C_2(s)$	R_1	R_2	Schedulable?	$idle(K)$	(I)	(II)	(VIII)
600	600	01	2.50	1.50	3.80	5.00	YES	31.20		Y	
	466	02	2.50	1.93	3.89	5.43	YES	30.68		Y	
	333	03	2.50	2.70	4.04	6.20	YES	29.76		Y	
	80	04	2.50	11.25	5.75	∅	NO	∅	Y	Y	
466	600	05	3.22	1.50	4.52	5.72	YES	29.76		Y	
	466	06	3.22	1.93	4.61	6.15	YES	29.24			
	333	07	3.22	2.70	4.76	6.92	YES	28.32		Y	
	80	08	3.22	11.25	6.47	∅	NO	∅	Y	Y	
333	600	09	4.50	1.50	5.80	7.00	YES	27.19		Y	
	466	10	4.50	1.93	5.89	7.44	YES	26.67		Y	
	333	11	4.50	2.70	6.05	8.21	YES	25.75			
	80	12	4.50	11.25	7.75	∅	NO	∅	Y		
80	600	13	18.75	1.50	20.05	∅	NO	∅			
	466	14	18.75	1.93	20.14	∅	NO	∅			Y
	333	15	18.75	2.70	20.29	∅	NO	∅			Y
	80	16	18.75	11.25	22.00	∅	NO	∅	Y		Y

Initially, consider the complete enumeration of all possible configurations $K \in \beta$ for this example. In this case, there are 4^2 possible configurations for this model. All these 16 configurations are listed in Table 2. If pruning step (I) is applied, frequency $80MHz$ is discarded for task $T2$, as its computational time would be $C_2(80) = 11.25$ and $D_2 = 10$, then $C_2(80) > D_2$. Hence, K's 04, 08, 12 and 16 are not evaluated if pruning step (I) is used. If pruning step (II) is executed, first $\beta' = \{01, 06, 11, 16\}$ is taken into consideration. Configuration 11 is the one in β' which is schedulable and provides the minimum idle time. So, 11 would be taken as the start point to evaluate remaining configurations in β. If pruning step (VIII) is considered during the enumeration of this task model, configurations 14, 15 and 16 would be not evaluated. At the point the enumeration reaches configuration 12, it will discover that 12 is not schedulable, and then, it will prune configurations 14, 15 and 16, because lowering frequency for task T_2 in this case will not give any other schedulable model. The resulting enumeration of β, if all pruning steps are used, contains only configurations 06, 11 and 13. For this task model, configuration 11 is the one which is schedulable and provides the minimum idle time. Configuration 01, which is the configuration with maximum frequency and voltage, would consume about $3398.64 \times C_l$, accordingly with Equation 1, considering C_l (circuitry capacitance) as a constant over time. Configuration 11, on the other hand, would consume about $1987.44 \times C_l$. Therefore, the optimum frequency set determined by the algorithm reduces about 41.52% of energy consumption, if compared to configuration 01.

6. Experimental Results

This is a *branch-and-bound* algorithm. Therefore, in the worst-case, even though all pruning steps are applied, all possible configurations $K \in \beta$ will be enumerated. The proposed algorithm performance evaluation is based on the number of enumerated configurations and on the time spent.

Table 3. Execution types

Type	Execution of
-	No type of prunings.
A	Step (I).
B	Step (II).
C	Step (VIII).
A+B	Steps (I) and (II).
A+C	Steps (I) and (VIII).
B+C	Steps (II) and (VIII).
A+B+C	Steps (I), (II) and (VIII).

All possible combination of pruning operations has been evaluated. In this case, there are eight

types of execution that are listed on Table 3.

For simplification purposes, the frequency and voltage switching overhead, σ from Equation 2, is considered to be zero in these experiments.

6.1. Case Study I

In this case study, there are six tasks in this model, they are executed in a system with four frequencies available and they share two resources. The set of available frequencies and voltages is listed in Table 4(a). The part of the task model with percentage of resource usage are shown in Table 4(b). The computational time spent during execution for each pruning combination is illustrated in Figure 3(a). The maximum time was 18 ms (no pruning) and the minimum time was 2 ms (pruning A+B+C). The number of evaluated configurations done during the enumeration for each pruning combination is illustrated in Figure 3(b).

Table 4. Case study I.

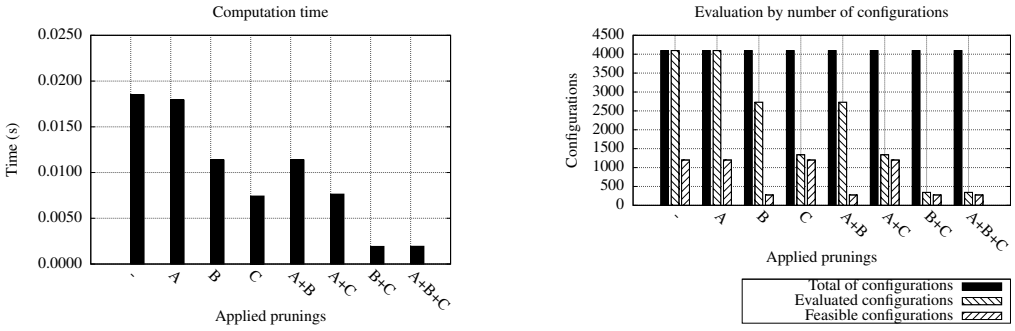
(a) Freq. and Voltages.

i	Frequency	Voltage
1	$440MHz$	$1.6V$
2	$120MHz$	$1.2V$
3	$60MHz$	$1.0V$
4	$30MHz$	$0.9V$

(b) Task model.

			Resource usage	
Task	Cycles	D_i	i	ii
1	2400	200	25%	0%
2	1200	50	13%	0%
3	1200	150	0%	30%
4	900	100	50%	0%
5	600	100	20%	0%
6	600	100	20%	10%

Results show that type A did not produce any pruning. Whilst types B and C have produced a considerable amount of prunings. This can be seen in the computation



(a) Computation time

(b) Number of evaluated configurations

Figure 3. Results for case study I.

time of Figure 3(a) and in the number of evaluated configurations of Figure 3(b), where type C has produced the lowest amount of time for execution and the lowest number of evaluated configurations. As presented in Figure 3(b), in order to determine the optimum frequency set, it was required to evaluate 361 of 4096 configurations, which

represents a reduction of 91.2% in the number of evaluated configurations. Considering the system executing under highest operating frequency, $440MHz$ under $1.6V$, from Equation (1), this model would consume $17664.00 \times C_l$. For this case study, the optimum frequency set is $\beta = \{60MHz, 440MHz, 30MHz, 120MHz, 440MHz, 440MHz\}$. Therefore, if the model uses the optimum configuration, it would consume $10812.00 \times C_l$, which represents a 38.79% of energy consumption reduction.

6.2. Case Study II

In this case study, there are twelve tasks in this model, they are executed in a system with four frequencies available and they share two resources. The set of available frequencies and voltages is listed in Table 5(a). The part of the task model with percentage of resource usage are shown in Table 5(b).

Table 5. Case study II.

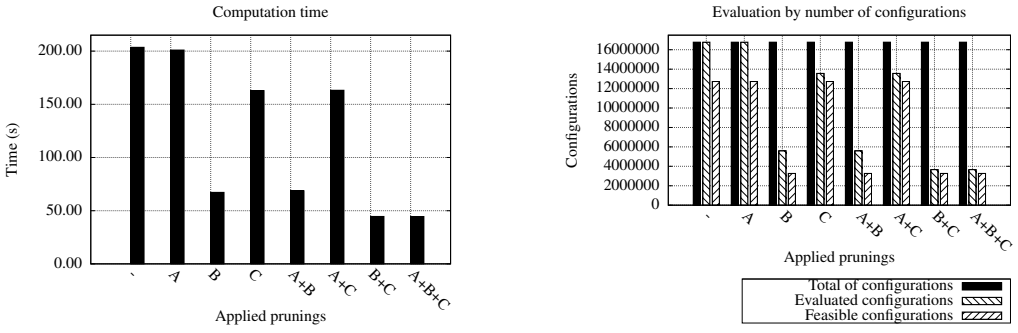
(a) Freq. and Voltages.

i	Frequency	Voltage
1	$300MHz$	$1.5V$
2	$250MHz$	$1.38V$
3	$220MHz$	$1.32V$
4	$150MHz$	$0.90V$

(b) Task model.

			Resource usage	
Task	Cycles	D_i	i	ii
1	180	10	13.4%	0%
2	180	20	44%	10%
3	180	30	0%	70%
4	180	30	0%	12%
5	180	30	12%	15%
6	180	30	0%	0%
7	180	30	13%	45%
8	120	10	3%	14%
9	120	20	4%	1%
10	120	30	0%	0%
11	120	30	0%	0%
12	100	10	0%	0%

The computational time spent during execution for each pruning combination of the proposed algorithm is illustrated in Figure 4(a). The maximum time was 210 s (no pruning) and the minimum time was 46 s (pruning A+B+C). The number of evaluated configurations done during the enumeration for each pruning combination is illustrated in Figure 4(b).



(a) Computation time

(b) Number of evaluated configurations

Figure 4. Results for case study II.

Results show that type B has produced best enumeration reduction. Type A has not produced any pruning. Reduction produced by type B has also appeared combined with other type of pruning. As presented in Figure 4(b), in order to determine the optimum frequency set, it was required to evaluate 3,662,613 of 16,777,216 configurations, which represents a reduction of $78,16\%$ in the number of evaluated configurations. Considering the system executing under highest operating frequency, $300MHz$ under $1.5V$, from Equation (1), this model would consume $4140.00 \times C_l$. In this case study, the optimum frequency set is $\beta = \{150MHz, 150MHz, 150MHz, 150MHz, 220MHz, 250MHz, 250MHz, 300MHz, 300MHz, 300MHz, 300MHz, 300MHz\}$. Therefore, if the model uses the optimum configuration, it would consume $2887.42 \times C_l$, which represent 30.26% in energy consumption reduction. This reduction is considerable and it would require only about tens of seconds during system design time in order to determine the optimum frequency set, as presented in Figure 4(a).

7. Conclusions and Future Works

This paper has shown a *branch-and-bound* algorithm to establish a set of optimum frequencies to be assigned to a real-time task model executed into a system with DVFS-enabled processor. The set of optimum frequencies is the one which produces lowest energy consumption and meets timing constraints from task model in such a way that provides the maximum utilization of idle times in each task. This work showed the response time schedulability analysis considering the specific frequency set.

The algorithm considers each frequency assignment combination as a configuration of a new model. A schedulability test is adopted in order to check each configuration feasibility. The algorithm itself consists of an enumerative search. Thus, in worst case it may reach unacceptable computation time. However, as presented in Section 6, the proposed pruning steps of this algorithm reduces considerably the amount of evaluated configurations and, therefore, the time spent for executing the algorithm. In the experiments, the number of evaluated configurations has been reduced to only 9% and 22% of total possible configurations, which is very a good result. Another important result is the amount of energy reduction when compared with executing tasks at high processor frequency. Our experimental results show that the proposed method may obtain energy reductions of 38.79%, and 30.46%.

For further research we intend: *(i)* to consider the energy needed to run a fraction of the tasks. If such amount of energy is already exceeding that needed to run a complete solution on a different branch, then the partial solution and its descendants can generally be pruned; *(ii)* to change the dynamic processor energy model to also include leakage current, memory access cost, I/O cost, and other energy overheads; and *(iii)* to develop a heuristic to produce a feasible configuration, maybe not optimum, but in a polynomial (non-exponential) computational time. *(iv)* to combine dynamic power management technique in our scheduling approach in order to reduce energy consumption by putting devices into sleep modes to optimize remaining idle time.

Acknowledgements

The authors would like to thank the partial financial support received from the Brazilian Council for Scientific and Technological Development (CNPq) through projects 554071/2006-1 and 575696/2008-7. And also for the partial financial support from the Nokia Corporation.

References

AbouGhazaleh, N., Mossé, D., Childers, B., Melhem, R., and Craven, M. (2003). Collaborative operating system and compiler power management for real-time applications. In *IEEE Real-Time Embedded Technology Applications Symposium*.

Azevedo, A., Issenin, I., Cornea, R., Gupta, R., Dutt, N., Veidenbaum, A., and Nicolau, A. (2002). Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework. In Design, Automation and Test in Europe Conference.

Choi, R. and Pedram, M. (2005). Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to onchip computation times. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.

Gutnik, V. (1996). Variable supply voltage for low power dsp. Master's thesis, Massachusetts Institute of Technology.

Havinga, S. (1997). A survey of energy saving techniques for mobile computers. Internal Report, University of Twente.

Mejia-alvarez, P., Levner, E., and Mossé, D. (2004). Adaptive scheduling server for power-aware real-time tasks. *ACM Transactions on Embedded Computing Systems*, 3(2):284–306.

Pillai, P. and Shin, K. G. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. pages 89–102.

Sha, L., Rajkumar, R., and Lehoczky, J. P. (1990). Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. on Computers*, 39:1175–1185.

Shin, D., Lee, S., and Kim, J. (2001). Intra-task voltage scheduling for low-energy hard real-time applications. In *IEEE Design & Test of Computers*.

Yao, F., Demers, A., and Shenker, S. (1995). A scheduling model for reduced cpu energy. In *IEEE Symposium on Foundations of Computer Science (FOCS'95)*.

Zhao, B. and Aydin, H. (2009). Minimizing expected energy consumption through optimal integration of dvs and dpm. In *International Conference on Computer-Aided Design (ICCAD'09)*, pages 449–456, New York, NY, USA. ACM.