Regulagem Dinâmica de Voltagem em Sistemas de Tempo Real

Bruno A. Novelli[1]**, J.C.B. Leite**[1]**, José M. Urriza**[2] **e Javier D. Orozco**[2]

[1]Instituto de Computação, Universidade Federal Fluminense
Niterói, Brasil

[2]Dep. Ingeniería Eléctrica y de Computadoras, Universidad Nacional del Sur
Bahía Blanca, Argentina

bnovelli@ic.uff.br, julius@ic.uff.br

jurriza@criba.edu.ar, jorozco@uns.edu.ar

Abstract. *This paper presents methods for power saving in real-time systems, through dynamic voltage scaling. These methods, for rate monotonic schedulers, use slack stealing to slowdown the frequency of execution of the processor. An off-line phase calculates the smaller frequency that still guarantees all timing constraints. An on-line phase takes into account the additional time produced by tasks that run for less than their worst-case execution time, and the slack left in the first phase, to yet reduce the frequency of execution. Results obtained through simulation show the improvements obtained with the use of the proposed methods.*

Resumo. *Neste trabalho são apresentados métodos para economia de energia em sistemas de tempo real, através do uso de regulagem dinâmica de voltagem. Os métodos, para escalonadores do tipo Rate Monotonic, fazem uso da determinação e adiantamento de folgas de execução. Uma etapa estática determina a menor freqüência de operação possível para o conjunto de tarefas, de tal forma que o escalonador ainda garanta todas as restrições temporais. Uma outra etapa, dinâmica, aproveita o tempo adicional gerado pela não utilização do tempo máximo de execução das tarefas, além da folga remanescente, para reduzir a freqüência de operação de cada uma delas. Resultados obtidos através de simulação demonstram significativa redução no consumo de energia.*

1. Introdução

Sistemas de tempo real, notadamente aqueles classificados como sistemas embutidos, têm incorporado novas características e, conseqüentemente, têm sido mais exigidos em termos de capacidade de processamento. Alguns desses sistemas são postos para operar em condições onde o suprimento de energia é limitado ou, ainda, fornecido por baterias. Infelizmente, o avanço da tecnologia de baterias tem sido lento em acompanhar as crescentes necessidades de consumo. Assim, restam duas opções: o uso de grandes baterias ou a realização de um gerenciamento de energia eficiente. Como o aumento físico da bateria nem sempre é possível ou desejável, técnicas de gerenciamento de energia que levem a um menor consumo tornam-se um importante requisito de projeto para esses sistemas. Além disso, como um sub-produto, deve ser notado que com um menor consumo a vida útil da bateria e dos dispositivos por ela alimentados é normalmente estendida. A redução da demanda energética tem ainda como vantagem a simplificação dos mecanismos de dissipação, muito útil quando as aplicações estão embutidas em dispositivos móveis, o aumento da confiabilidade dos componentes e uma redução em seu custo. Esse trabalho contribui para a universalidade da computação na medida em que seus resultados podem ser aplicados a dispositivos portáteis (e.g., PDAs, celulares), mais e mais frequentes nas sociedades modernas, cujo emprego depende, em grande parte, de mecanismos de economia de energia.

Dentre os diversos tipos de componentes de *hardware* que constituem um dispositivo móvel destacam-se o processador, o sistema de memória e a interface de rede como os principais consumidores de energia. Para cada um deles existem técnicas de *software* que têm por objetivo minimizar esse consumo. O estudo aqui apresentado é focado no processador, pois é usualmente o componente que mais consome energia no sistema. Isto é verdadeiro não somente para pequenos dispositivos de mão como PDAs [Ellis, 1999], que tem poucos componentes, mas também para computadores portáteis [Lorch e Smith, 1998] que possuem muitos componentes. Neste trabalho é utilizada a técnica conhecida como RDV - Regulagem Dinâmica de Voltagem (DVS - *Dynamic Voltage Scaling* [Weiser et al., 1994, Bertini e Leite, 2004]), em conjunto com um algoritmo para determinação de folgas de processamento (tempo ocioso), para o escalonamento de tarefas de tempo real, com o objetivo de economia de energia.

A principal área de interesse desse trabalho são os sistemas de tempo real que, por definição, são aqueles submetidos a requisitos de natureza temporal. Nestes sistemas, os resultados devem estar corretos não somente do ponto de vista lógico-aritmético, mas também devem ser gerados no momento correto. Esses sistemas podem ser classificados como críticos (*hard real-time*) e não críticos (*soft real-time*). Nos sistemas críticos um atraso ou resposta incorreta é considerado um erro inaceitável, que pode resultar em uma catástrofe ou envolver perda de vidas (e.g., sistemas de controle de tráfego aéreo, de controle de linhas ferroviárias e de usina nucleares). Já nos sistemas não críticos pode-se aceitar ocasionalmente uma resposta atrasada (e.g., sistemas de telefonia digital ou de vídeo conferência). Quando um sistema cumpre todas as restrições de tempo diz-se que ele é escalonável. Embora o presente trabalho seja voltado para sistemas críticos, a mesma técnica pode ser utilizada, com algumas modificações, para sistemas não críticos, como os empregados em aplicações multimídia (e.g., [Yuan e Nahrstedt, 2003]).

Seguindo o modelo multiprocessos-monoprocessador do trabalho seminal de Liu e Layland [Liu e Layland, 1973], as tarefas de tempo real a serem consideradas nesse trabalho são críticas, periódicas, independentes e preemptáveis. O conjunto de tarefas é especificado como S(n) = {(C_1, T_1, D_1), ... , (C_n, T_n, D_n)}, onde C_i, T_i e D_i denotam, respectivamente, o tempo de execução de pior caso (WCET - *Worst Case Execution*

Time), o período de ativação e o prazo para cada tarefa *i.* Uma suposição comum é que, ∀i, $D_i = T_i$. Quando as tarefas competem pelo processador o conflito é resolvido de acordo com as regras definidas por uma política de prioridades. É chamado de instante crítico aquele no qual todas as tarefas, simultâneamente, irão competir pelo processador. O hiperperíodo de um conjunto de tarefas é definido como o mínimo múltiplo comum dos seus períodos. Durante um hiperperíodo há momentos em que o processador estará ocupado executando tarefas e outros nos quais o processador se encontrará ocioso. Estes espaços vazios (ou seja, onde há ociosidade do processador) são chamados de folga (*slack*).

No trabalho aqui apresentado utiliza-se um método de roubo de folga (*slack stealing*) para adiantar partes do tempo livre e, assim, poder diminuir a voltagem de alimentação e a freqüência de operação do processador, garantindo ainda o atendimento aos prazos definidos pelas aplicações. Outros trabalhos na área de economia de energia combinam somente a redução da voltagem e da freqüência através de cálculos realizados ainda em tempo de projeto e de modificações dinâmicas na grade de execução. Esse último caso consiste em aproveitar o tempo não utilizado por uma tarefa (denominado *gain time*) para diminuir a velocidade de execução de tarefas subseqüentes. Isso é possível pois o tempo de execução de uma tarefa, em aplicações de tempo real, pode variar bastante em relação ao seu WCET. Por exemplo, [Ernst e Ye, 1997] relatam que o tempo de pior caso pode chegar a ser até 10 vezes maior que o melhor tempo de execução. Deve ser enfatizado que esses trabalhos não tiram vantagem da possibilidade de avançar folgas. No trabalho aqui apresentado, além do potencial aproveitamento do *gain time*, uma economia de energia adicional é obtida através do mecanismo de avanço de folgas.

Esse artigo é organizado como se segue. Na seção 2 é feita uma introdução ao funcionamento do mecanismo de RDV. Na seção 3 são apresentados trabalhos relacionados. Na seção 4 é explicado o método proposto nesse trabalho para economia de energia. Na seção 5 são apresentados alguns resultados e, finalmente, na seção 6 é feita a conclusão.

2. Regulagem Dinâmica de Voltagem

A tecnologia CMOS é hoje comumente utilizada em diversos processadores. Em circuitos integrados implementados com essa tecnologia o consumo de energia *(E)* é aproximadamente proporcional ao quadrado da voltagem *(V)* e varia linearmente com a freqüência ($\frac{E}{clock} \propto V^2$). Contudo, a redução da freqüência operacional do processador, isoladamente, não leva a uma redução de consumo, pois implica no aumento do tempo de execução das tarefas. Deve ser notado que, para circuitos integrados reais, admitindo que para uma dada voltagem é sempre utilizada a freqüência máxima correspondente a esse nível, a redução na tensão de alimentação exige uma conseqüente redução na freqüência de operação. No que se segue, quando houver menção a redução de freqüência estará subentendido que haverá uma correspondente diminuição na tensão de alimentação.

As figuras 1 e 2 ilustram o funcionamento da técnica de RDV. Nessas figuras, as setas claras delimitam o período da tarefa e *D* representa o prazo máximo de execução. A figura 1 apresenta a execução de uma tarefa sem utilização de RDV. Nessa figura a tarefa executa à voltagem e freqüência máximas, requerendo *C* unidades de tempo e tendo uma certa quantidade de folga. Na figura 2 é mostrada a execução da mesma tarefa com utilização de RDV. Nesse caso, a tarefa executa em uma voltagem *v'*, menor que a voltagem máxima, requerendo assim mais unidades de tempo para executar o mesmo número de ciclos, diminuindo a folga e também a quantidade de energia gasta na sua execução.

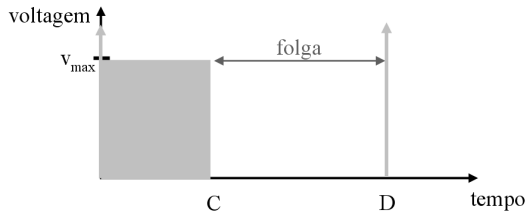

Figura 1: Execução de tarefa sem utilização de RDV

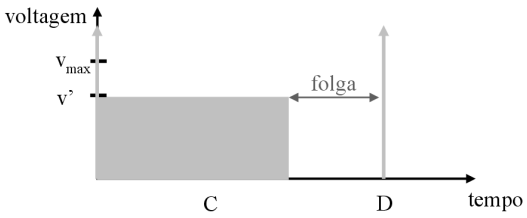

Figura 2: Execução de tarefa com utilização de RDV

Vários processadores comerciais (e.g., AMD *Mobile*, Intel *Centrino Mobile* e *Xscale Technology*, Transmeta *Efficeon* e *Crusoe*) implementam o mecanismo de regulagem dinâmica de voltagem (RDV). Para todos os processadores citados há um número limitado de níveis de tensão que podem ser utilizados. Ou seja, a tecnologia atual ainda não permite o oferecimento de circuitos com variação contínua de seu ponto de operação. Com base no mecanismo de RDV diversos algoritmos têm sido desenvolvidos, especialmente para sistemas de tempo real ([Weiser et al., 1994, Pering e Brodersen, 1998, Pillai e Shin, 2001, Aydin et al., 2001a, Kim et al., 2002b, Urriza et al., 2004]).

3. Trabalhos Relacionados

Diversos trabalhos têm tratado da aplicação de RDV e de algoritmos de escalonamento de processos para obtenção de economia de energia. Em sua grande maioria, esses trabalhos assumem a hipótese de que o consumo de energia é diminuído quando há redução da tensão de alimentação. Contudo, isso nem sempre é verdadeiro. Em [Miyoshi et al., 2002] é demonstrado que, para alguns processadores comerciais, algumas das freqüências de operação são energeticamente ineficientes, ou seja, a freqüência imediatamente superior produz melhores resultados. Para que a hipótese anteriormente indicada seja verdadeira, o processador deve ter uma relação potência-freqüência convexa não decrescente. O trabalho desenvolvido por [Saewong e Rajkumar, 2003] indica como obter uma grade ótima de freqüências de operação de forma a minimizar os erros de quantização causados pelo uso de freqüências discretas. Esse é um resultado importante, não só para a construção e avaliação de modelos teóricos, mas também para a construção de processadores reais.

Nos trabalhos que utilizam RDV os períodos de ociosidade são tradicionalmente aproveitados, nos sistemas de tempo real, com o objetivo de minimizar o consumo de energia no sistema mas garantindo o prazo. Pode-se associar a esses períodos a execução de tarefas aperiódicas não críticas ([Lehoczky e Ramos-Thuel, 1992, Sprunt et al., 1989, Strosnider et al., 1995]) ou ainda a execução de partes opcionais de tarefas críticas cujo processamento possa implicar em algum tipo de recompensa ([Aydin et al., 2001b, Santos et al., 2002, Aydin et al., 2004]).

Em [Ishihara e Yasuura, 1998] são apresentados alguns resultados teóricos para sistemas que usam RDV. Dentre esses, pode-se ressaltar:

- *Teorema 1*: Se um processador somente pode utilizar um número discreto de voltagens, o escalonamento com o uso de no máximo dois níveis de tensão, para qualquer restrição temporal, minimiza o consumo de energia;
- *Teorema 2*: Se um processador somente pode utilizar um número discreto de voltagens, as duas voltagens que minimizam o consumo de energia, para uma dada restrição temporal, são aquelas imediatamente vizinhas à voltagem ideal (aquela que implicaria em um único nível para todas as tarefas).

Um trabalho baseado em *gain time* e aproveitamento de folga é discutido em [Pillai e Shin, 2001], onde os autores apresentam abordagens estáticas (*off-line*) e dinâmicas (*on-line*) para sistemas de tempo real críticos. Na abordagem estática seleciona-se a menor freqüência operacional possível que ainda permita ao escalonador, no caso EDF (*Earliest Deadline First*) [Liu e Layland, 1973] ou RM (*Rate Monotonic*) [Liu e Layland, 1973], garantir todos os prazos do conjunto de tarefas. A redução é feita com base nos testes de escalonabilidade conhecidos para esses dois algoritmos, pela introdução de um fator de escala (relação entre a freqüência de operação pretendida e a freqüência máxima permitida pelo processador) para os tempos de execução máximos das tarefas. Na abordagem dinâmica, ou seja, para o caso em que as tarefas executam por menos tempo que o seu WCET, são apresentados dois algoritmos: o *Cycle Conserving*, desenvolvido para escalonadores RM e EDF, e o *Look Ahead*, desenvolvido somente para EDF. Ambos os algoritmos se baseiam no cálculo da utilização do processador ao término da execução de cada tarefa, de forma a reclamar quaisquer ciclos (previstos no WCET) não utilizados. A diferença entre esses algoritmos é que no *Look Ahead* é assumida uma abordagem otimista, mais agressiva, onde, no início da execução da tarefa ela é processada a uma velocidade inferior, podendo essa velocidade (freqüência-voltagem) ser aumentada para garantir o atendimento à restrição temporal.

Em [Kim et al., 2002a] é apresentado um algoritmo baseado em *gain time* para um conjunto de tarefas periódicas, em sistemas de tempo real críticos, utilizando um escalonador EDF. A principal característica deste algoritmo está na determinação do *gain time*, que é feita de duas formas. A primeira determina o *gain time* para as tarefas de maior prioridade que completam sua execução antes do seu WCET. Já na segunda forma, o aproveitamento provém das tarefas de menor prioridade que durante sua execução utilizaram menos que o seu WCET. Ao serem preemptadas, elas permitem que tarefas de maior prioridade possam aproveitar esse *gain time*. Este método acrescenta um pequeno *overhead* para a determinação do *gain time*, porém não há nenhuma fase *off-line*.

Em [Kim et al., 2002b], os autores apresentam uma avaliação de vários algoritmos com base em RDV para conjuntos de tarefas periódicas e premptivas em sistemas de tempo real críticos. Na avaliação é utilizado um simulador que incorpora as características desses diferentes esquemas. Como resultado, eles apresentam uma comparação da eficiência na conservação de energia entre os algoritmos baseados em EDF ([Pillai e Shin, 2001, Aydin et al., 2001a, Kim et al., 2002a, Shin et al., 2000]) e aqueles que usam RM ([Pillai e Shin, 2001, Shin et al., 2000]) com um limite inferior obtido teoricamente. Os autores concluem o trabalho mostrando que alguns dos algoritmos que utilizam EDF atingem resultados próximos ao limite inferior (cerca de 9 a 12% piores). Para os algoritmos baseados em RM eles reportam resultados que ficam a uma distância significativa desse limite.

O trabalho discutido em [Aydin et al., 2004] também apresenta um algoritmo baseado em RDV para o escalonamento de tarefas periódicas em sistemas de tempo real críticos. A solução apresentada é feita em duas fases. A primeira, *off-line*, onde é obtida uma freqüência ótima para cada tarefa, assumindo que elas gastam seu tempo máximo de processamento em cada ativação. Na segunda fase, *on-line*, o tempo não gasto pelas tarefas (em relação ao seu WCET) é aproveitado durante a execução. É reportado no artigo que uma economia de energia de até 50% é obtida nessa fase em relação à fase estática. É ainda apresentado um mecanismo adaptativo de ajuste de velocidade que utiliza a informação da carga de trabalho média (medida) para prever términos antecipados de execuções futuras e, assim, reduzir especulativamente velocidades. Segundo os autores, essa fase consegue atingir uma economia suplementar de energia de até 20% sobre o es-

quema dinâmico anteriormente citado. Os resultados apresentados nesse artigo são para escalonadores com prioridades dinâmicas.

4. O Método Proposto

O escalonamento com prioridades fixas é o método mais utilizado atualmente para a implementação de aplicações sujeitas a restrições de tempo real. No entanto, nos diversos métodos de economia de energia encontrados na literatura o escalonamento é majoritariamente realizado usando-se prioridades dinâmicas, mais especificamente, o algoritmo EDF. Algoritmos baseados em prioridades dinâmicas, como EDF e LLF [Mok, 1983], são mais eficientes no aproveitamento do processador que aqueles baseados em prioridades fixas, como o RM, sendo o seu limite teórico de utilização igual a 100%. Esses algoritmos são capazes de tratar tanto tarefas aperiódicas quanto periódicas, podendo ser usados em aplicações de tempo real críticas e não críticas. Contudo, se as aplicações necessitam somente de tarefas periódicas, o algoritmo RM, por sua implementação eficiente, simples e intuitiva, mostra-se uma opção interessante. Adicionalmente, RM é um algoritmo muito utilizado em sistemas industriais e foi indicado pelo DoD dos EUA para aplicações de tempo real críticas [Obenza, 1993]. Uma razão para isso é o fato do EDF ter um comportamento instável em circunstâncias especiais, como, por exemplo, em condições de sobrecarga [Buttazzo, 2003]. A grande aceitação de RM como base para os sistemas operacionais de tempo real atuais foi um importante motivador para a sua utilização nesse trabalho.

4.1. Fase 1: Escala Estática de Freqüências

Dado um conjunto de tarefas, definido para uma determinada fase de operação do sistema, a pior situação de carga que pode ser apresentada para o escalonador é quando ocorre o instante crítico e quando todas as tarefas executam o WCET. O instante crítico é definido como aquele no qual todas as tarefas do sistema estão simultâneamente prontas para serem executadas. Nessa situação, o objetivo é escolher a menor freqüência de operação possível para o conjunto de tarefas, de tal forma que o escalonador RM ainda garanta todas as restrições de tempo. Nessa fase, o método aqui apresentado funciona como uma variante ao proposto para o algoritmo RM em [Pillai e Shin, 2001], procurando utilizar a folga disponível que ocorre para o sistema operando em freqüência máxima. A determinação da folga existente é feita pelo mesmo algoritmo a ser utilizado na Fase 2 (ver seção 4.2.1). Ao fim da fase estática, são conhecidas as folgas para todas as tarefas, no início do hiperperíodo.

Essa primeira fase já oferece uma substancial melhoria na utilização da energia. Contudo, a economia é limitada pela quantidade de níveis de voltagem e freqüência do processador e pelo fator de utilização do sistema. Como os modelos de processadores reais ainda não trabalham com variações contínuas no mecanismo de RDV, nem sempre é possível utilizar uma freqüência ótima e é, portanto, selecionado o nível discreto imediatamente acima ao que seria o ótimo para o RM. Em relação ao fator de utilização, essa fase não aproveita de forma ótima os tempos ociosos do processador (ver exemplo a seguir) sendo, portanto, utilizada uma fase dinâmica que, além de aproveitar a folga remanescente, utiliza também o *gain time*.

Sejam as tarefas da tabela 1. Admitindo-se uma situação de instante crítico, a figura 3(a) apresenta a escala gerada pelo algoritmo RM para o hiperperíodo desse conjunto de tarefas, assumindo-se freqüência máxima de operação. Nessa figura, os números nos retângulos indicam as tarefas e *e* representa os períodos ociosos do processador. No instante inicial, as folgas para as tarefas 1, 2 e 3 valem, respectivamente, 2, 1 e 1.

É possível notar que, sem a utilização de RDV, as tarefas executam à máxima freqüência e por 75% do tempo do hiperperíodo. Existirá, então, 25% de folga a cada hiperperíodo. Em 3(b) está representada a escala ótima em termos de ocupação de períodos ociosos, desde que seja possível ao processador assumir qualquer freqüência e voltagem. Em 3(c) está representada a escala gerada pela aplicação da fase estática do algoritmo, onde todas as tarefas executam à mesma freqüência (menor que a máxima). Como pode ser observado, mesmo após essa fase, ainda restam períodos de folga. Comparando-se as figuras 3(b) e 3(c) vê-se que há possibilidade de otimização do mecanismo, o que será feito em sua fase dinâmica. Graficamente, dado que a energia é calculada como o produto da potência pelo tempo, pode-se visualizar o ganho obtido como a área limitada entre a linha horizontal de f_{max} e o topo dos retângulos indicativos de execução das tarefas.

Tabela 1: Conjunto de tarefas periódicas

Tarefa	WCET	Período
1	1	3
2	1	4
3	1	6

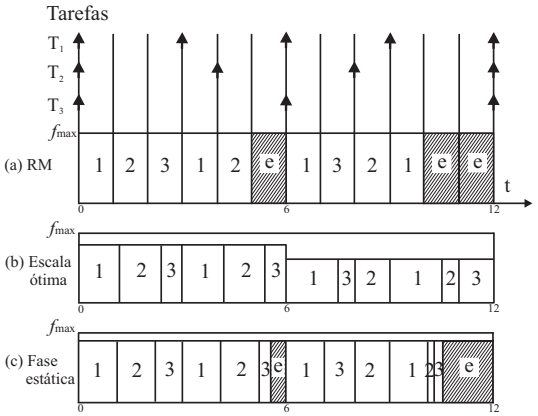

Figura 3: Escalonamento do conjunto de tarefas da Tabela 1

4.2. Fase 2: Escala Dinâmica de Freqüências

Embora os conjuntos de tarefas de tempo real crítico sejam especificados com base em seu WCET, essas tarefas, durante sua execução, utilizam, em média, um tempo bem menor. Cada vez que uma tarefa está para ser executada não há como saber quanto tempo ela gastará e, para manter a escalonabilidade do conjunto, deve-se considerar que ela utilizará o seu tempo máximo.

4.2.1. Determinação da Folga

Nessa fase, precisa-se determinar a quantidade de folga disponível para que se possa reduzir a freqüência de operação de cada tarefa individualmente. O método aqui apresentado é aquele desenvolvido por Urriza e Orozco [Urriza e Orozco, 2004].

É aqui assumido que, em um tempo t qualquer, os seguintes dados estão disponíveis para o sistema operacional:

$l_i(t) \Rightarrow$ tempo no qual a tarefa i teve sua última liberação para ser executada
$x_i(t) \Rightarrow$ próxima liberação da tarefa i; $x_i(t) = l_i(t) + p_i$

$d_i(t) \Rightarrow$ vencimento da próxima invocação da tarefa i; $d_i(t) = x_i(t) + p_i$
$ta \Rightarrow$ tempo atual
$wa(t) \Rightarrow$ tempo total de execução em relação ao início do hiperperíodo
$S_i(t) \Rightarrow$ folga disponível para a tarefa i
$c_i \Rightarrow$ tempo de execução da tarefa i
$tresc_i \Rightarrow$ tempo restante de execução da tarefa i no instante atual
$p_i \Rightarrow$ período da tarefa i
$\epsilon \Rightarrow$ variável auxiliar
$hp_i \Rightarrow$ conjunto das tarefas de maior prioridade que a tarefa i

Notar que $l_i(t)$, $x_i(t)$, $d_i(t)$ são calculados no tempo t. Caso t seja 0, $\forall_i$, $x_i(t) = 0$.

O método de determinação de folga objetiva determinar a quantidade máxima de folga, $S_i(t)$, que pode ser roubada da tarefa i durante o intervalo $[t, t + d_i(t)]$, de tal forma que o escalonador RM ainda garanta todas as restrições de tempo. Este método utiliza contadores de folga individuais para cada tarefa. Qualquer período de tempo ocioso entre o término da execução da tarefa i e o seu prazo poderá ser trocado com a execução de i sem que esta perca seu prazo. Contudo, a quantidade máxima de folga que pode ser roubada é igual ao total do período ocioso do intervalo acima indicado. Para garantir que a tarefa i manterá seu prazo, considera-se o tempo de execução de pior caso da tarefa atual e das demais tarefas a serem executadas. Assume-se também que todas tarefas do conjunto hp_i serão re-invocadas tão cedo quanto possível na próxima liberação da tarefa i e subseqüentemente a cada período. O algoritmo para cálculo de folga está indicado através do pseudo-código apresentado adiante.

Ao fim da execução de uma tarefa (ou seja, quando o escalonador é chamado): (i) a folga por ela utilizada deve ser subtraída das folgas de todas as outras tarefas (há somente atualização de contadores - $O(n)$); (ii) caso essa tarefa tenha produzido um *gain time*, ele é acrescentado à folga de todas as tarefas de menor prioridade (também aqui há somente atualização de contadores); e, (iii) a folga para a próxima ativação dessa tarefa é então calculada (é somente nesse momento que se utiliza o algoritmo de determinação de folga - $O(n^2)$). Caso, ao fim dessa execução haja um período ocioso, o valor desse período é subtraído de todas as folgas (novamente, só atualização de contadores).

4.2.2. Distribuição de Folga

No início da execução de uma tarefa é verificada a quantidade de folga para ela disponível, cujo valor será igual à menor folga existente para todas as tarefas. Para a distribuição dessa folga é então utilizada uma heurística gulosa. Essa escolha foi feita em virtude de sua baixa complexidade de implementação e também pelo fato de que, em muitas aplicações, o tempo de execução de uma tarefa pode ser bem menor que o seu WCET [Ernst e Ye, 1997]. Deve ser notado que, em sistemas que apresentam muito *gain time*, a heurística gulosa apresenta melhores resultados pois, desde o início da execução, de forma otimista, todo o *gain time* que vai surgindo é aproveitado. A seguir são descritos outros refinamentos que, em conjunto com a heurística gulosa para distribuição de folga, permitem melhorias suplementares na economia de energia.

- *Execução em Um Nível:* Esse método utiliza toda a folga disponível para verificar se é possível baixar a freqüência de operação da tarefa atual. Caso seja possível, ela executará nessa nova freqüência e a folga utilizada será subtraída da folga disponível. Como os processadores comerciais que permitem RDV possuem um número discreto de níveis de voltagem, à tarefa atual, na maioria das vezes, será atribuído um nível imediatamente acima daquele que seria o ideal. Caso a folga disponível não seja suficiente

Algoritmo 1 - Algoritmo para determinação da folga

1: **Se** Primeira tarefa **Então**
2: $S_i(t) \leftarrow d_i(t) - ta - c_1$
3: **Senão**
4: $\epsilon \leftarrow 0,00000001$;
5: $S_i(t) \leftarrow 0$;
6: $wa(t) \leftarrow \sum_{\forall\ j\ \in\ hp(i)\ \cup\ i} \left(\left\lfloor \frac{ta}{p_j} \right\rfloor * c_j - tresc_j \right)$
7: **Para** j $\leftarrow$ 1 até i **Faça**
8: $aux \leftarrow \left\lfloor \frac{d_i(t)}{p_j} \right\rfloor * p_j$;
9: **Se** $(Abs(aux - (d_i(t))) < \epsilon)$ **e** $(j \neq i)$ **Então**
10: $aux \leftarrow aux - p_j$
11: **Fim Se**
12: $auxfolga \leftarrow aux - ta - \left(\sum_{j=1}^{i} \left\lceil \frac{aux}{p_j} \right\rceil * c_j \right) + \ wa(t)$
13: **Se** $(Abs(S_i(t) - auxfolga) > \epsilon)$ **e** $(S_i(t) - auxfolga < \epsilon)$ **Então**
14: $S_i(t) \leftarrow auxfolga$
15: **Fim Se**
16: **Fim Para**
17: **Fim Se**
18: Retorne $S_i(t)$

para se baixar ao menos um nível a tarefa executará normalmente na freqüência em que se encontra e a folga poderá ser alocada para a próxima tarefa. A figura 2 ilustra a execução de uma tarefa em um nível de voltagem e freqüência. Notar que esse método representa a simples implementação da heurística gulosa.

- *Execução em Dois Níveis:* Um refinamento pode ser feito com base nos teoremas apresentados em [Ishihara e Yasuura, 1998] e citados na seção 3. Aqui, uma vez encontrado o nível ótimo, a tarefa será executada utilizando-se os níveis imediatamente abaixo e acima. A fim de aproveitar variações no tempo de execução das tarefas, e de forma otimista, as tarefas serão inicialmente executadas no menor dos dois níveis de voltagem. A figura 4 ilustra a utilização de dois níveis de voltagem/freqüência durante a execução de uma tarefa.

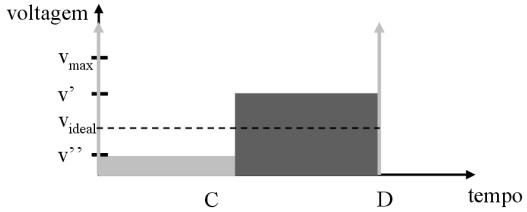

Figura 4: Execução da tarefa em dois níveis de voltagem

Durante a execução, uma tarefa pode ser pre-emptada. Antes disso, ao início de sua execução, um determinado ponto de operação do processador já havia sido determinado. Durante a pre-empção novas folgas podem surgir e serão, automaticamente, incorporadas à folga da tarefa que foi pre-emptada. Assim, quando ela retorna a executar, um novo ponto de operação será calculado, permitindo, eventualmente, uma economia de energia adicional.

5. Resultados

Nesta seção são apresentados e discutidos alguns resultados obtidos através de simulação para permitir avaliar a eficiência dos métodos desenvolvidos. Nas simulações realizadas foram comparados os resultados com o valor ótimo. O cálculo desse valor é feito admitindo-se que o processador possa assumir qualquer freqüência, e calcula-se então qual o valor de freqüência que leva a 100% de utilização do processador. Esta consideração é equivalente ao escalonamento com EDF utilizando um esquema contínuo de freqüências. Essa será a freqüência ótima para efeito de comparação entre os métodos de economia de energia aqui apresentados.

As simulações foram realizadas com conjuntos de 4, 8, 12 e 16 tarefas. Esses conjuntos foram gerados de tal forma a não serem harmônicos e também a garantir que todas as tarefas possuam períodos distintos. Cada tarefa tem uma probabilidade uniforme de possuir um período pequeno, médio ou grande. Nos experimentos realizados as faixas desses períodos são, respectivamente, $(3 - 10)$, $(11 - 30)$ e $(31 - 100)$, e cada período é gerado com base em uma distribuição uniforme no intervalo escolhido. O WCET de uma tarefa é gerado aleatoriamente, também seguindo uma distribuição uniforme, entre $0,1$ e o período da tarefa. De forma a obter o fator de utilização desejado, os tempos de execução são modificados por um fator de escala.

Para o conjunto de resultados aqui apresentados é utilizado o processador teórico que está definido na tabela 2. Esta tabela é utilizada para calcular a quantidade de energia gasta pelos métodos. A utilização do processador teórico é justificada já que os processadores reais mais novos não informam os valores necessários ao cálculo do consumo de energia. Finalmente, é assumido que o consumo de energia no estado ocioso (*idle*) é igual ao consumo no menor nível de voltagem/freqüência.

Tabela 2: Processador teórico

Nível	1	2	3	4	5	6	7	8	9	10
Freqüência	100	200	300	400	500	600	700	800	900	1000
Voltagem	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
Corrente	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1

Na simulação, para efeito de avaliação da fase dinâmica, o tempo de execução das tarefas é gerado aleatoriamente. Duas distribuições são utilizadas, uma uniforme e outra exponencial. Nas figuras 5 a 9, a marca de 0,2 no eixo das abscissas indica que os tempos de execução das tarefas variam entre 20% e 100% do WCET, para o caso da distribuição uniforme. Para os gráficos com distribuição exponencial a marca de 0,2 indica que o valor médio do tempo de execução das tarefas é de 20% do WCET. Em todos eles, cada ponto foi gerado executando-se o hiperperíodo 10 vezes e é fornecido com intervalo de confiança de 95%. Finalmente, em todas as figuras referidas, no eixo das ordenadas é fornecida a energia consumida como um percentual do pior caso (ou seja, 100%, que corresponderia à utilização de voltagem/freqüência máximas em todo o hiperperíodo).

Embora as figuras 5, 6, 7 e 8 diferenciem-se pelas quantidades de tarefas, as curvas mostradas nos gráficos são muito semelhantes. Em todos eles, o fator de utilização afeta substancialmente o consumo de energia, como esperado, mas o número de tarefas tem quase nenhum impacto. Esse resultado é consistente com o obtido por [Pillai e Shin, 2001]. Adicionalmente, deve ser notado que a distribuição do tempo de execução das tarefas pode ter influência nos resultados. No caso da distribuição exponencial um número no eixo das abscissas indica o valor médio do tempo de execução, e no da distribuição uniforme indica o mínimo. Assim, a primeira produzirá muito mais *gain time* e, conseqüentemente, maior economia de energia. Contudo, é interessante notar que isso é mantido mesmo quando compara-se um ponto com 60% de WCET no primeiro caso com um de 20% nos gráficos com distribuição uniforme. Essa observação é importante para a proposição de novas heurísticas para a distribuição de folga, talvez mais agressivas, quando se conhece a distribuição do tempo de execução. Pelos motivos acima expostos, na seqüência somente serão apresentados resultados para o conjunto de oito tarefas e com a distribuição uniforme (já que ela apresenta uma situação de pior caso).

Considerando agora as figuras 9(a), 9(b), 9(c) e 6, que indicam resultados para diferentes fatores de utilização (20%, 40%, 60% e 80%), nota-se que o esquema em dois níveis, na pior situação, ficará a 9% (para utilização de 80%) acima do ótimo e no melhor caso, 0,8% superior (utilização de 20%). Observa-se que os métodos afastam-se do ótimo à medida que se aumenta o fator de utilização, o que é natural já que há diminuição da folga. Finalmente, como esperado, nota-se que os resultados produzidos pelo método de 2 níveis convergem para aqueles do método de 1 nível à medida que diminui a variabilidade do tempo de execução, já que há uma conseqüente diminuição das folgas. Da mesma forma, para qualquer método, a energia consumida aumenta quando a variabilidade do tempo de execução em relação ao WCET diminue.

Para efeito de motivação será agora apresentada uma comparação entre o algoritmo *Cycle-conserving RM* (ccRM) desenvolvido por [Pillai e Shin, 2001] e os métodos de execução em um nível e dois níveis aqui propostos. O conjunto de tarefas utilizado é aquele indicado no artigo acima citado (ver tabela 3). O modelo de processador utilizado é o teórico indicado naquele trabalho (3 níveis: 50%, 75% e 100% da freqüência máxima) e é assumido que o consumo de energia no estado ocioso é zero.

Tabela 3: Conjunto de tarefas do artigo [Pillai e Shin, 2001]

Tarefa	Período	WCET	Ativação 1	Ativação 2
1	8	3	2	1
2	10	3	1	1
3	14	1	1	1

Admitindo a situação de instante crítico, a figura 10(a) apresenta a escala gerada pelo algoritmo RM para os dois primeiros períodos da tarefa 1, assumindo-se o WCET para todas as tarefas. Nota-se que, sem utilização de RDV, as tarefas executam à máxima freqüência e ocupam 87,5% do tempo considerado. Existirá, então, 12,5% de folga que poderá ser utilizado através de RDV.

Na figura 10(b) (figura 5(f) de [Pillai e Shin, 2001]) está representada a escala gerada pela aplicação do algoritmo ccRM, para as duas primeiras ativações das tarefas, assumindo-se variação do tempo de execução como indicado na tabela 3, colunas Ativação 1 e Ativação 2. Simplificadamente, esse algoritmo consiste em utilizar a folga existente entre o instante em que a tarefa inicia execução e o próximo prazo (considerando-se

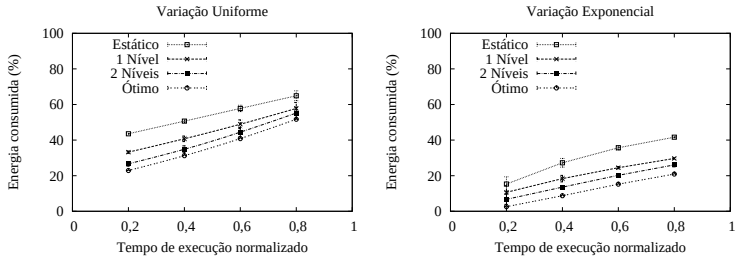

Figura 5: Conjunto de 4 tarefas com 80% de fator de utilização

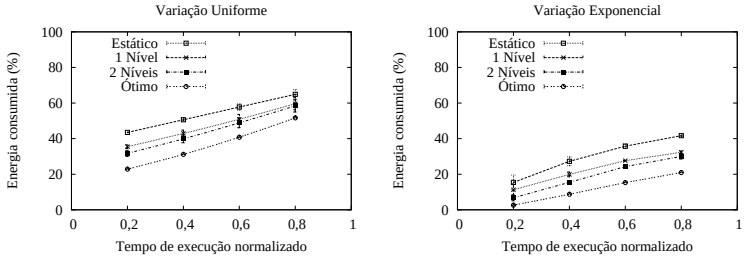

Figura 6: Conjunto de 8 tarefas com 80% de fator de utilização

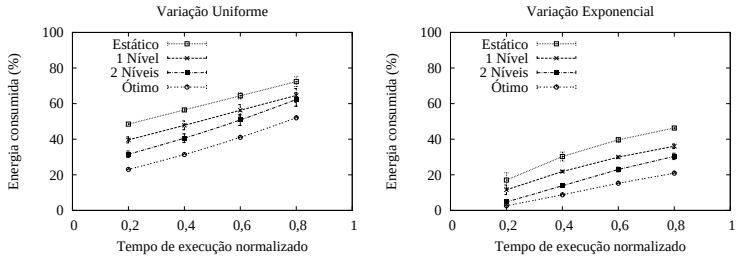

Figura 7: Conjunto de 12 tarefas com 80% de fator de utilização

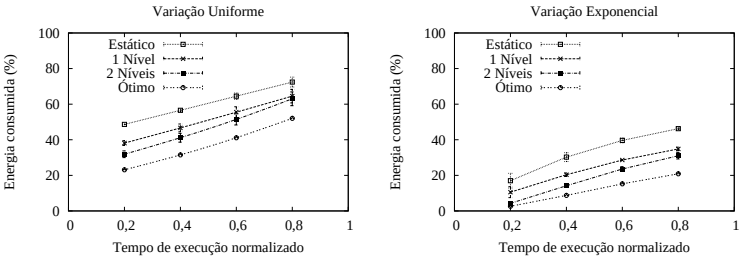

Figura 8: Conjunto de 16 tarefas com 80% de fator de utilização

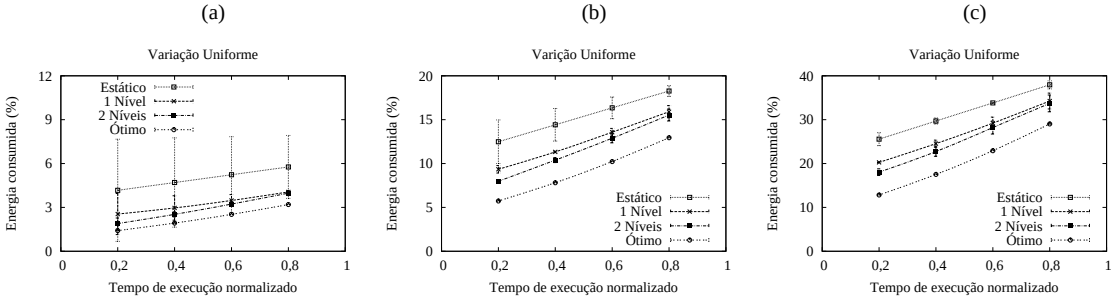

Figura 9: Conjunto de 8 tarefas com: (a) 20%; (b) 40%; e, (c) 60% de fator de utilização

todas tarefas ativas). Nota-se nessa figura que o processador ficou ocioso cerca de 38% do tempo. Nas figuras 10(c) e 10(d) estão apresentadas as escalas geradas pelos métodos de execução em um nível e em dois níveis descritos anteriormente. Nesses casos, respectivamente, houve redução para cerca de 22% e 16,6% do tempo ocioso do processador. Analisando o consumo total de energia nesses dois períodos de execução da tarefa 1, verifica-se que o método de um nível produz uma economia de 11,4% em relação ao ccRM e que o método de dois níveis economiza 13,4% relativamente àquele algoritmo. Esses métodos apresentam melhores resultados pois utilizam melhor a folga disponível e, portanto, executam as tarefas em níveis de voltagem e freqüência mais baixos.

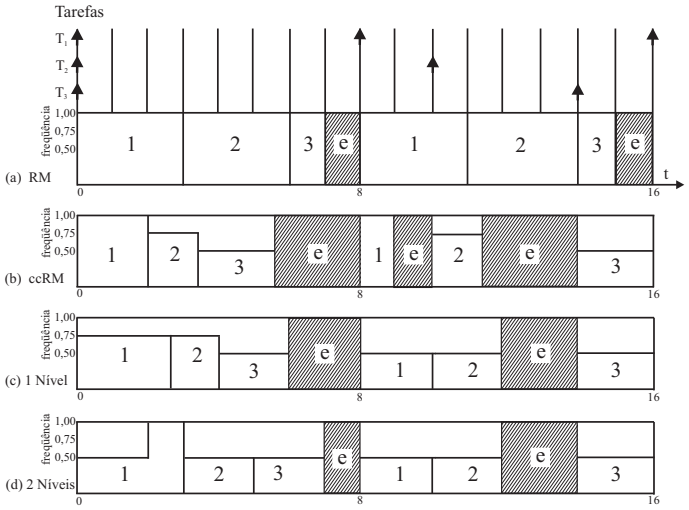

Figura 10: Escalonamento do conjunto de tarefas da Tabela 3

Admitindo-se agora o mesmo modelo utilizado no experimento indicado na figura 6, com variação uniforme, a figura 11 fornece uma comparação entre os métodos ccRM, um nível e dois níveis. Para tempos de execução tendo como valor mínimo 20% do WCET, os métodos de 1 e 2 níveis produzem resultados melhores em 8,3% e 17,8%, respectivamente, que o consumo de energia do ccRM. Já na situação em que o tempo normalizado é 0,8, as melhorias são reduzidas para 2,3% e 4,3%, respectivamente. Isso é esperado em função da diminuição da quantidade de *gain time* no sistema. Esse último caso é interessante pois, como há pouco *gain time*, efetivamente é mostrada a eficiência dos métodos na utilização de adiantamento de folga. Deve ser ressaltado que o modelo de processador utilizado nesse experimento, com maior número de níveis em relação ao experimento anterior, pode influenciar nos resultados, tanto na fase estática quanto na dinâmica. Essa investigação, contudo, só será feita em trabalhos futuros já em planejamento.

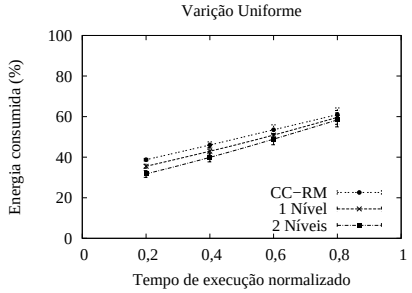

Figura 11: Comparação com algoritmo de Pillai & Shin

6. Conclusão

Dispositivos portáteis têm se tornado cada vez mais populares. Uma de suas principais características é a de depender de alimentação por bateria para a sua operação. Adicionalmente, em várias situações, o tipo de carga a ser processada apresenta requisitos de tempo real. Dessa forma, técnicas que permitem economia de energia vêm sendo pesquisadas. Um dos maiores consumidores de energia em sistemas como os aqui citados é o processador. Neste trabalho são apresentados métodos de escalonamento de tarefas de tempo real que procuram ocupar os tempos livres (folga e *gain time*) da UCP, colocando-a a operar em nível de voltagem (e também de freqüência de relógio) mais baixo, de forma a diminuir o consumo de energia. Os métodos apresentados forneceram bons resultados nos diversos experimentos realizados e, em particular, quando comparados a um dos principais métodos para escalonadores RM publicados na literatura. Esses resultados são promissores e demonstram significativa redução no consumo de energia. Novos testes estão planejados para melhor avaliar o impacto de diversos parametros, como, por exemplo, o número de níveis de voltagem do processador e os valores relativos desses níveis. Finalmente, deve-se acrescentar que o uso do método de 2 níveis de execução por tarefa pode acarretar uma intervenção adicional do sistema operacional. Esse impacto e sua eventual minimização também serão objeto de estudos futuros.

7. Agradecimentos

Os autores agradecem ao CNPq e ao CONICET pelo financiamento parcial deste trabalho.

Referências

Aydin, H., Mejía-Alvarez, P., Mossé, D., e Melhem, R. (2001a). Dynamic and aggressive scheduling techniques for power-aware real-time systems. In *22nd IEEE Real-Time Systems Symposium*, p. 95–105, Londres, Reino Unido.

Aydin, H., Mejía-Alvarez, P., Mossé, D., e Melhem, R. (2001b). Optimal reward-based scheduling for periodic real-time tasks. *IEEE Transactions on Computers*, 50(1):111–130.

Aydin, H., Mejía-Alvarez, P., Mossé, D., e Melhem, R. (2004). Power-aware scheduling for periodic real-time tasks. *IEEE Trans. on Computers*, 53(5):584–600.

Bertini, L. e Leite, J. C. B. (2004). Um breve survey: Escalonamento em sistemas de tempo real com otimização do consumo de energia. In *VI Workshop de Tempo Real*, p. 288–297, Gramado, RS, Brasil.

Buttazzo, G. C. (2003). Rate monotonic vs. edf: Judgment day. In *3rd International Conference on Embedded Software*, p. 67–83, Filadélfia, PA, EUA.

Ellis, C. S. (1999). The case for higher-level power management. In *7th IEEE Workshop on Hot Topics in Operating Systems*, p. 162–167, Rio Rico, Arizona, EUA.

Ernst, R. e Ye, W. (1997). Embedded program timing analysis based on path clustering and architecture classification. In *International Conference on Computer-Aided Design*, p. 598–604, San Jose, California, EUA.

Ishihara, T. e Yasuura, H. (1998). Voltage scheduling problem for dynamically variable voltage processors. In *International Symposium on Low-Power Electronics and Desing*, p. 197–201, Monterey, California, EUA.

Kim, W., Kim, J., e Min, S. (2002a). A dynamic voltage scaling algorithm for dynamic-priority hard real-time systems using slack time analysis. In *Conference on Design, Automation and Test in Europe*, p. 788–794, Washington, DC, EUA.

Kim, W., Shin, D., Yun, H., Kim, J., e Min, S. (2002b). Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *8th IEEE Real-Time and Embedded Technology and Applications Symposium*, p. 219–228, San Jose, California, EUA.

Lehoczky, J. P. e Ramos-Thuel, S. (1992). An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *IEEE Real-Time Systems Symposium*, p. 110–123, Phoenix, Arizona, EUA.

Liu, C. L. e Layland, J. W. (1973). Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1):46–61.

Lorch, J. R. e Smith, A. J. (1998). Apple macintosh energy consumption. *IEEE Micro*, 18(6):54–63.

Miyoshi, A., Lefurgy, C., Hensbergen, E. V., Rajamony, R., e Rajkumar, R. (2002). Critical power slope: Understanding the runtime effects of frequency scaling. In *16th ACM International Conference on Supercomputing*, p. 35–44, Nova York, Nova York, EUA.

Mok, A. K.-L. (1983). *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. PhD thesis, MIT.

Obenza, R. (1993). Rate monotonic analysis for real-time systems. *IEEE Computer*, 26(3):73–74.

Pering, T. e Brodersen, R. (1998). Energy efficient voltage scheduling for real-time operating systems. In *4th IEEE Real-Time Technology and Applications Symposium*, Denver, Colorado, EUA. work-in-progress session.

Pillai, P. e Shin, K. G. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th Symposium on Operating Systems Principles*, p. 89–102, Banff, Alberta, Canadá.

Saewong, S. e Rajkumar, R. (2003). Practical voltage-scaling for fixed-priority rt-systems. In *9th IEEE Real-Time and Embedded Technology and Applications Symposium*, p. 106–115, Toronto, Canadá.

Santos, R. M., Urriza, J., Santos, J., e Orozco, J. (2002). Heuristic use of singularities for on-line scheduling of real-time mandatory/reward-based optional systems. In *14th Euromicro Conference on Real-Time Systems*, p. 103–110, Vienna, Áustria.

Shin, Y., Choi, K., e Sakurai, T. (2000). Power optimization of real-time embedded systems on variable speed processors. In *International Conference on Computer-Aided Design*, p. 365–368, San Jose, California, EUA.

Sprunt, B., Sha, L., e Lehoczky, J. P. (1989). Aperiodic task scheduling for hard real-time systems. *The Journal of Real-Time Systems*, 1(1):27–60.

Strosnider, J. K., Lehoczky, J. P., e Sha, L. (1995). The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Trans. on Computers*, 44(1):73–91.

Urriza, J. M., Novelli, B. A., Leite, J. C. B., e Orozco, J. D. (2004). Economia de energia em dispositivos móveis. In *VI Workshop de Comunicação sem Fio e Computação Móvel*, p. 48–56, Fortaleza, CE, Brasil.

Urriza, J. M. e Orozco, J. D. (2004). Métodos rápidos para el cálculo del slack stealing exacto y aproximado para aplicaciones en sistemas embebidos. Relatório técnico, Dep. de Ing. Eléctrica y de Computadoras, Universidad Nacional del Sur, Bahía Blanca, Argentina.

Weiser, M., Welch, B., Demers, A., e Shenker, S. (1994). Scheduling for reduced cpu energy. In *1st Symposium on Operating Systems Design and Implementation*, p. 13–23, Monterey, California, EUA.

Yuan, W. e Nahrstedt, K. (2003). Energy-efficient soft real-time cpu scheduling for mobile multimedia systems. In *20th Symposium on Operating Systems Principles*, p. 149–163, Bolton Landing, Nova York, EUA.