Real-Time Dynamic Voltage Scaling for the EPOS Operating System

Gustavo Nardon Meira and Antônio Augusto Fröhlich
Software/Hardware Integration Lab (LISHA)
Federal University of Santa Catarina (UFSC)
Florianópolis, Brazil
Email: {meira,guto}@lisha.ufsc.br

Arliones Hoeller Jr.
Automation and Systems Engineering Department (DAS)
Federal University of Santa Catarina (UFSC)
Florianópolis, Brazil
Email: arliones@das.ufsc.br

***Abstract*—Several implementations of Real-Time Dynamic Voltage and Frequency Scaling (RT-DVFS) have been made in the last decade. Most of them, however, are adaptations performed over some Linux-based operating system. The real-time support on Linux systems depend on complex modifications of the system kernel and is often not regarded as hard real-time support. This work presents the design and implementation of the DVFS support for the hard real-time schedulers of EPOS (Embedded Parallel Operating System) – an application-driven operating system designed to support embedded applications. The design presented here supports the insertion of RT-DVFS heuristics into the system schedulers in a loosely-coupled fashion. The design is implemented in EPOS and evaluated in a XScale processor. Effectiveness of the RT-DVFS heuristic is assessed by implementing two classic RT-DVFS algorithms that were proposed as extensions to the Earliest Deadline First scheduler.**

I. INTRODUÇÃO

Dispositivos embarcados alimentados por bateria trazem consigo restrições conflitantes: autonomia e desempenho. Ao mesmo tempo que devem economizar energia, ganham a responsabilidade sobre tarefas inteligentes que necessitam de processamento poderoso. Mesmo com a exigência de alto desempenho, o pico da demanda de computação costuma acontecer apenas em alguns momentos do funcionamento de sistemas de tempo real [1]. Em sistemas dedicados, onde se tem um maior conhecimento sobre o comportamento das aplicações, tem-se encontrado um palco interessante para a aplicação de técnicas de DVFS em sistemas de tempo real, também chamadas de RT-DVFS.

Grande parte dos estudos em RT-DVFS são realizados através de simulações, que possuem resultados restritos, devido a fatores imprevisíveis ou não modelados [2]. Dos trabalhos que desenvolvem implementações reais de suporte a RT-DVFS, comumente as demonstrações ocorrem em sistemas Linux com a adição de módulos ou extensões [1], [3], [4], [2]. O sistema operacional EPOS, onde realizamos nossa implementação, já possui suporte nativo a tarefas de tempo real [5]. Além disso, o EPOS é um sistema operacional que segue a metologia ADESD (*Application-Driven Embedded System Design*) [6], o que benefecia a proposta configurável e adaptável apresentada neste trabalho devido à organização orientada a objetos do sistema e à sua natureza não-monolítica.

Neste trabalho, utiliza-se o porte do EPOS para a plataforma PXA255, um processador Intel XScale com suporte a DVFS amplamente utilizado. As heurísticas para RT-DVFS *Static Voltage Scaling* e *Cycle-conserving* para o escalonador EDF [1] são utilizadas como estudo de caso para avaliação do ambiente criado.

O restante do paper está organizado do seguinte modo. A seção II descreve o ambiente experimental utilizado no trabalho. A seção III descreve os algoritmos RT-DVFS implementados para avaliar o sistema. A seção IV realiza uma breve introdução ao sistema EPOS e mostra o funcionamento de seu ambiente de tempo real. Em sequência, a seção V descreve o projeto e implementação da extensão RT-DVFS do EPOS. A seção VI mostra os resultados dos experimentos conduzidos. Finalmente, a seção VII conclui o trabalho.

II. AMBIENTE EXPERIMENTAL

O processador PXA255 é uma plataforma de hardware destinada a aplicações que exigem alto desempenho e baixo consumo de energia [7]. O processador possui suporte a alteração dinâmica de frequência e tensão para três dispositivos: núcleo do processador, barramento do sistema (*PXBus*) e memória SDRAM. A relação entre os principais componentes do PXA255 pode ser observada na figura 1. Na mesma figura, em destaque, encontram-se os dispositivos que suportam DVFS.

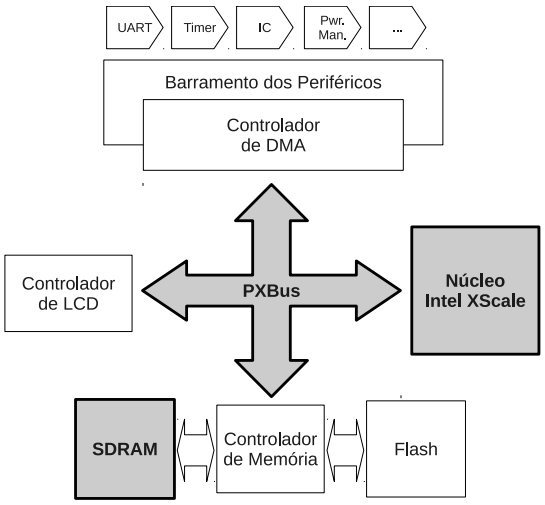

Figure 1. Topologia de componentes do PXA255.

Segundo o *Intel PXA255 Developer's Manual* [7], existe um conjunto de configurações de frequência válidas para estes dispositivos. Desse conjunto, para os fins deste trabalho, selecionamos apenas as configurações para as quais não há variação na frequência da SDRAM. Este subconjunto é apresentado na tabela I. Embora a frequência da SDRAM permaneça constante, as variações de frequência do PXBus continuam fazendo sentido ao passo que o controlador DMA utiliza o mesmo barramento para transferência de dados entre periféricos e a memória principal ou entre perirféricos.

Tensão do processador (V)	Frequência do Núcleo (MHz)	Frequência do PXBus (MHz)	Frequência da SDRAM (MHz)
1.0	99,5	50	99,5
1.0	199,1	99,5	99,5
1.1	298,6	99,5	99,5
1.3	398,1	186	99,5

Table I
SUBCONJUNTO DE CONFIGURAÇÕES DO PXA255 UTILIZADAS NESTE TRABALHO.

III. ALGORITMOS *Real-Time DVFS*

Algoritmos de *Real-Time Dynamic Voltage (and Frequency) Scaling*, ou simplesmente RT-DVFS, são os algoritmos de escalonamento de tempo-real capazes de gerenciar a capacidade de computação dos processadores (frequência) para reduzir o consumo de energia enquanto continuam garantindo que os prazos das tarefas sejam atendidos. O termo aparece primeiramente no trabalho de Pillai et al. [1], que apresentou adaptações aos escalonadores Earliest Deadline First (EDF) e Rate Monotonic (RM) para suportar DVFS. Neste trabalho, os escalonadores clássicos apresentados por Pillai são utilizados para demostrar o funcionamento da implementação RT-DVFS realizada.

Já que nem sempre as tarefas de uma aplicação de tempo real acabam executando como seu pior caso (C_i), e a utilização do processador nem sempre é total, é possível usufruir da economia energética oferecida pelas técnicas de DVFS em sistemas de tempo real. Por exemplo, se uma tarefa termina sua execução mais cedo que o esperado pode-se utilizar a folga gerada para que uma próxima tarefa seja executada com uma frequência mais baixa. Esta técnica também é conhecida como *slack reclamation* [8]. Ainda assim, esse tipo de decisão deve ser tomada com cautela, de modo que nenhum prazo seja perdido.

O trabalho original de Pillai propõe três heurísticas: alteração de tensão estática (*Static Voltage Scaling*), conservação de ciclos (*Cycle-conserving RT-DVFS*), e preditiva (*Look-ahead RT-DVFS*). Das classes citadas, este trabalho utiliza em sua implementação as duas primeiras em conjunto ao algoritmo de escalonamento EDF. Esta escolha é justificada pela simplicidade de implementação e, ao mesmo tempo, por permitir a comparação entre heurísticas que tomam decisões estaticamente e heurísticas que tomam decisões durante a execução do sistema. O funcionamento das duas heurísticas escolhidas é explicado a seguir.

A. Static Voltage Scaling *para EDF (svsEDF)*

O svsEDF é um mecanismo simples que se beneficia da alteração de tensão e frequência, mantendo a execução das tarefas dentro de seus respectivos prazos. Como o próprio nome da técnica sugere, esta é uma configuração estática da frequência do processador. A ideia é obter a menor frequência possível de modo que, mesmo neste baixo desempenho, o conjunto de tarefas satisfaça o teste de escalonabilidade do escalonador EDF. A frequência, por ser configurada estaticamente, só é alterada se o conjunto de tarefas é alterado. Assim sendo, se durante a execução, uma nova tarefa é adicionada ao sistema, o conjunto de tarefas está sendo alterado, então é necessária a escolha de uma nova frequência de funcionamento.

É possível perceber que, alterando a frequência de operação por um fator α $(0 < \alpha \leq 1)$, será necessário alterar os piores tempos de computação para cada tarefa em um fator $1/\alpha$. Observe que os períodos e deadlines mantêm-se inalterados. Com um novo pior tempo de computação, haverá alteração no teste de escalonamento.

Para verificar a escalonabilidade dos conjuntos de tarefas pelo EDF utiliza-se o teste de escalonabilidade baseado em utilização. Este teste, agora com a frequência alterada pelo fator α, resulta em uma proporcionalização da utilização máxima (originalmente igual a 1) de acordo com α[1], representada como $\sum_{i=1}^{n} \frac{C_i}{P_i} \leq \alpha$, onde C_i e P_i são, respectivamente, o tempo de computação e o período da tarefa i, em um sistema com n tarefas.

A consciência energética da heurística *Static Voltage Scaling* para EDF consiste então na otimização do conjunto das m frequências possíveis para o processador, sendo escolhida a menor delas capaz de manter o conjunto de tarefas escalonável segundo o teste apresentado. Se o conjunto de tarefas passa no teste de escalonabilidade para uma nova frequência e, ainda, as tarefas também não ultrapassam seu novo pior caso de resposta, este mecanismo assegura que os prazos não serão comprometidos. Se o teste de escalonabilidade falha para todas configurações possíveis, então não é possível escalonar o conjunto de tarefas.

Pela seleção da frequência ser realizada anteriormente à execução do conjunto de tarefas, o novo algoritmo torna-se fracamente acoplado ao escalonador de tempo real [1]. Isto significa que uma configuração prévia do sistema pode ser feita e, em tempo de execução, um escalonador EDF comum pode ser empregado. Por outro lado, este mecanismo não oferece economia de energia nos momentos onde uma tarefa não utiliza seu pior tempo de computação, o que pode acontecer com frequência em sistemas reais. Para este caso, algoritmos que levam em conta a utilização real da tarefa,

[1]O novo teste de escalonabilidade nada mais é do que a aplicação do fator $\frac{1}{\alpha}$ a cada C_i no teste clássico do EDF. Multiplicando os dois lados da inequação por α, obtém-se a expressão apresentada aqui.

como o exposto adiante, conseguem um melhor redução de consumo de energia.

B. Cycle-conserving RT-DVFS *para EDF (ccEDF)*

Uma maneira de tirar maior proveito do uso de DVFS durante o escalonamento EDF é aproveitar as folgas de tempo obtidas quando tarefas não atingem seu pior tempo de computação. Para isso, o algoritmo ccEDF assume que a tarefa inicialmente ocupa seu pior tempo de resposta mas, ao seu término, verifica qual foi o real uso do processador na execução, identificando as folgas. O algoritmo então leva as folgas em consideração para ajustar o desempenho do processador, reduzindo o consumo de energia ao alterar a frequência de operação durante a execução.

Na heurística *Cycle-conserving RT-DVFS* para EDF, três eventos do sistema devem ser capturados: a alteração do conjunto de tarefas, o início da execução de uma tarefa e a conclusão de uma tarefa. Nestes eventos, as seguintes ações são executadas:

- **Na alteração do conjunto de tarefas** é possível comportar-se como na heurística estática apresentada anteriormente. Isto garante que a configuração inicial é válida para o sistema, sem nenhum prejuízo, já que o svsEDF é baseada em um cenário de pior caso.
- **No início da execução de cada tarefa** calcula-se a utilização do sistema levando em consideração o pior caso da tarefa que está para iniciar, mas o caso real das tarefas que já executaram.
- **Na conclusão de cada tarefa** registra-se o tempo real de execução da tarefa para informar o escalonador do surgimento da folga.

A figura 2 demonstra a heurística *Cycle-conserving RT-DVFS* para EDF sendo aplicada. No exemplo, os valores possíveis para α são $1,0$ ou $0,75$ ou $0,50$. O conjunto de tarefas da tabela II é utilizado como exemplo. A tabela III mostra o tempo de resposta destas tarefas com $\alpha = 1$, para a primeira e segunda invocação de cada tarefa. As setas apontam a utilização total calculada no início da execução da tarefa (utilizando-se dados estáticos), e também a utilização total calculada com os dados obtidos *online*, ao final da execução das tarefas. Sempre que a utilização calculada fica abaixo de um valor possível de α, a frequência do sistema é modificada para este valor.

Tarefa (T_i)	Pior Tempo de Computação (C_i)	Período (P_i)	Utilização da tarefa (U_i)
T_1	$3ms$	$8ms$	$0,375$
T_2	$3ms$	$10ms$	$0,300$
T_3	$1ms$	$14ms$	$0,071$

Table II
EXEMPLO DE CONJUNTO DE TAREFAS

Tarefa	Tempo de Computação na Primeira Invocação	Tempo de Computação na Segunda Invocação
T_1	$2ms$	$1ms$
T_2	$1ms$	$1ms$
T_3	$1ms$	$1ms$

Table III
TEMPO DE COMPUTAÇÃO PARA CADA INVOCAÇÃO DO EXEMPLO.

IV. ESCALONAMENTO DE TEMPO REAL NO EPOS

O EPOS (*Embedded Parallel Operating System*) [6] é um sistema operacional dirigido a aplicações embarcadas de alto desempenho. Ele segue a metodologia ADESD (*Application-Driven Embedded System Design*), utilizando técnicas de orientação a objetos, orientação a aspectos e programação estática em busca de se adequar às restrições presentes nas aplicações para o qual é utilizado. Em sua maior parte, o sistema é desenvolvido na linguagem C++.

Para manter a portabilidade do sistema operacional, entidades chamadas *mediadores de hardware* [9] fornecem interfaces simples para acesso a funções dependentes de máquina. Estas interfaces são utilizadas por entidades mais abstratas do sistema, chamadas de *abstrações*. Alguns exemplos de abstrações do EPOS são `Thread` e `Scheduler`, que utilizam mediadores de hardware como, por exemplo, o `Timer` e `CPU`.

A abstração `Periodic_Thread` do EPOS é responsável por oferecer suporte à execução de tarefas periódicas de tempo real no sistema. A partir dessa abstração, é possível criar um objeto ao qual são associados um período e uma rotina a ser executada. Este período é representado pelo atributo `period` da classe `Alarm` (Figura 3). A rotina associada à tarefa periódica é representada na mesma imagem pelo atributo `entry`, herdado por `Periodic_Thread` da classe `Thread`.

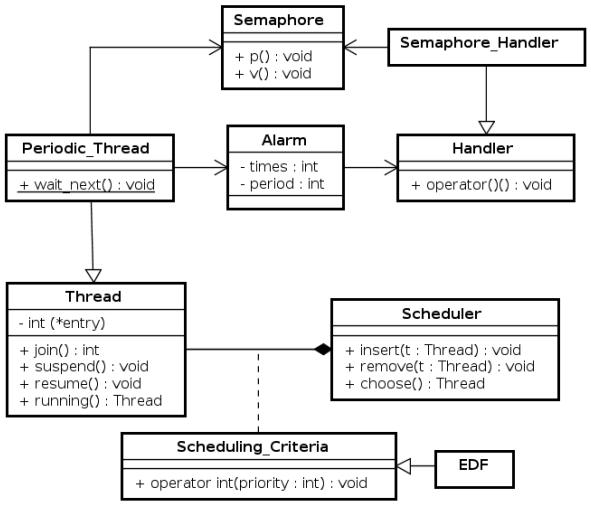

Figure 3. Diagrama de classes com a abstração `Periodic_Thread`

A rotina especificada pelo parâmetro `entry` deve ser a tarefa periódica da aplicação. Esta rotina deve ser explicitamente programada como um laço de repetição e, a cada iteração, o método `Periodic_Thread::wait_next`

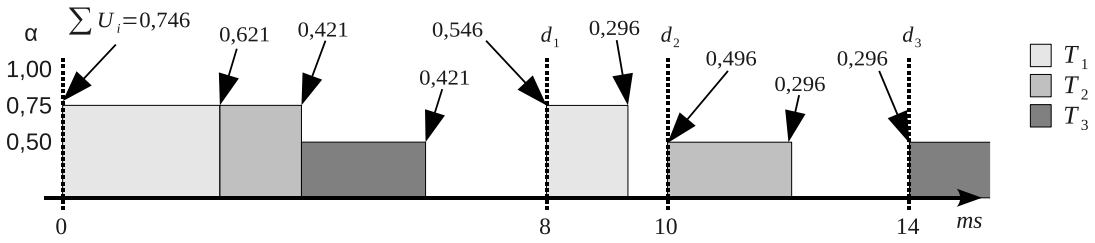

Figure 2. Exemplo de escalonamento *Cycle-conserving RT-DVFS* para EDF com a utilização calculada em cada início e término de tarefa.

deve ser chamado pelo programador. Esta chamada, através da operação `p` sobre o semáforo associado à abstração `Periodic_Thread`, faz com que a thread em execução seja suspensa. Esta thread ficará suspensa, i.e. não utilizará a CPU, até que a operação `v` seja realizada sobre o mesmo semáforo. Para oferecer o comportamento periódico às tarefas, a abstração `Alarm` é então utilizada.

A abstração `Alarm` invoca periodicamente uma instância da classe `Handler` (apresentada na Figura 3), neste caso especializado para executar a operação `v` sobre o semáforo que bloqueia a thread periódica. A thread pode, então, voltar à fila de prontas e, eventualmente, ser escalonada segundo os critérios do escalonador. Ao retomar a execução, ela volta ao ponto onde invocou `Periodic_Thread::wait_next`, obedecendo ao comportamento do laço escrito na aplicação.

V. Solução RT-DVFS para o Epos

Para implementar as heurísticas RT-DVFS, o escalonador precisa estar ciente de eventos referentes às tarefas do sistema. Estes são eventos como inicio e término de instâncias de tarefas (*jobs*), ou até mesmo alterações no conjunto de tarefas. Para que fosse possível atribuir ações a estes eventos, algumas modificações foram necessárias no modelo de threads periódicas do EPOS. Estas modificações são apresentadas na Figura 4.

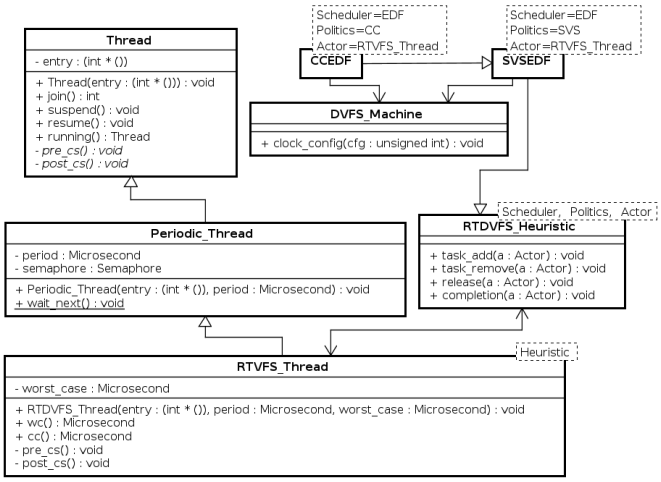

Figure 4. Diagrama de classes da solução `RTVFS_Thread` do Epos

A. Captura de Início e Término de instâncias de uma Tarefa

Os eventos de início e término de instâncias de tarefas são capturados através da reimplementação do método `wait_next` para a `RTDVFS_Thread`. O diagrama de sequência da Figura 5 mostra o momento em que estes eventos são reportados à heurística, ou seja, antes e depois da aquisição das operações com o semáforo da thread periódica.

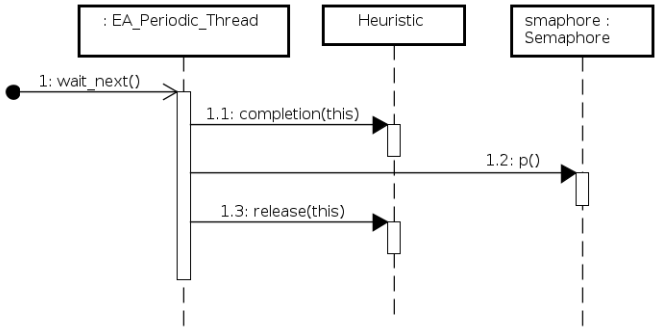

Figure 5. Diagrama de sequência demonstrando o comportamento do método `wait_next` da classe `RTDVFS_Thread`.

B. Captura de Modificações no Conjunto de Tarefas

A reação à alteração no conjunto de tarefas é dada de duas maneiras: adição ou remoção de tarefas. Para capturar o evento de adição, em toda construção de uma nova `RTDVFS_Thread`, a classe `RTDVFS_Heuristic` tem o método `task_add` invocado, sendo passada como parâmetro para sua execução a instância de `RTDVFS_Thread` em ação. Analogamente, na destruição de uma `RTDVFS_Thread`, o método `task_remove` é chamado.

C. Adição de Heurísticas

Como estudo de caso, duas heurísticas RT-DVFS foram selecionadas para a implementação utilizando o suporte criado: *Static Voltage Scaling* e *Cycle-conserving* [1]. Ambas foram utilizadas em conjunto com o escalonador EDF já presente no Epos. A política *Static Voltage Scaling* se baseia em dados de pior caso do tempo de execução das tarefas, assim sendo, ela calcula estaticamente a configuração de frequência ideal para o pior caso de execução das tarefas. A política *Cycle-conserving* se baseia em dados capturados de forma *on-line*, ajustando a

configuração de frequência e tensão do processador durante a execução de acordo com o uso real que as tarefas promovem no sistema.

A implementação das heurísticas foram realizadas utilizando especialização de *templates* em C++. A escolha da heurística pode ser programada estaticamente, como uma configuração do sistema operacional, sendo que cada heurística é especializada através de três parâmetros:

- `Scheduler`: resultados neste trabalho são para o Earliest Deadline First, mas o Rate Monotonic presente no EPOS também pode ser utilizado;
- `Politics`: a política a ser utilizada: *Static Voltage Scaling* ou *Cycle-conserving*);
- `Actor`: a abstração que utiliza a heurística, neste trabalho `RTDVFS_Thread`.

VI. Experimentos e Resultados

O ambiente experimental foi criado com base em um conjunto pré-definido de três tarefas periódicas, com 300, 400 e 500 milisegundos de período, de modo que estas possuíssem uma utilização real configurável através de um laço com um número arbitrário de iterações. A potência média foi escolhida como métrica de avaliação por abstrair as variáveis que caracterizam a potência dinâmica do sistema, que são, além da tensão (V), a frequência (f), a capacitância (C) e o fator de atividade (a) do circuito[2]. A tensão e a frequência variarão conforme a escolha da configuração do sistema pelo escalonador, e a capacitância e taxa de atividade dependerão da carga do sistema. Como os instantes em que estas variáveis se alteram no sistema dependem do algoritmo de escalonamento não é adequado avaliá-las isoladamente. E já que o tempo para executar as tarefas no sistema pode variar, utilizar consumo de energia como métrica também pode gerar valores inadequados[3]. Contudo, o consumo de energia total ainda pode ser obtido utilizando a potência média: $E = P_M \times \Delta t$.

A plataforma de hardware utilizada foi uma placa-mãe Gumstix Connex, que possui um PXA255 como módulo de processamento. Juntamente a esta placa, uma expansão chamada Gumstix HWUART foi utilizada para a comunicação com um PC para a carga de aplicações do EPOS e depuração.

O gráfico apresentado na Figura 6 mostra a potência média obtida para uma utilização real igual à utilização de pior caso, i.e., toda tarefa realmente utiliza a quantidade de tempo estimada como de pior caso de execução. Observe que este gráfico é capaz de mostrar a relação entre potência média do sistema e as configurações apresentadas na tabela I, já que neste cenário de pior caso, as heurísticas tendem a utilizar as configurações correspondentes à utilização do sistema. Por exemplo, se as tarefas promovem uma utilização real de 0,6, o sistema possivelmente trabalhará a 298,6MHz (0,75 da maior frequência possível).

O gráfico apresentado na Figura 7 mostra o comportamento das heurísticas quando a utilização real das tarefas do sistema

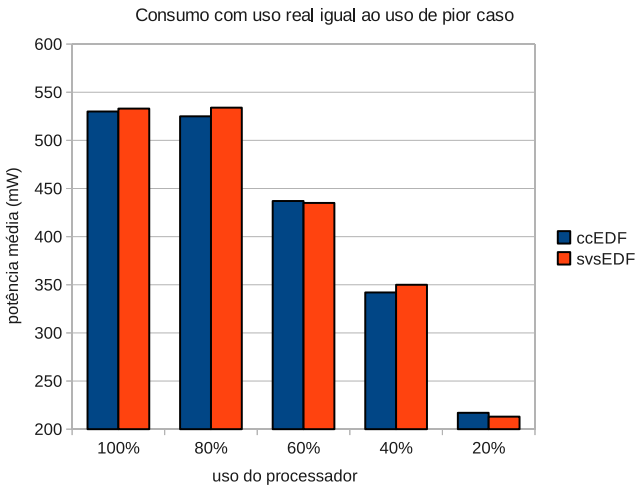

Figure 6. Potência média com utilização real igual à de pior caso.

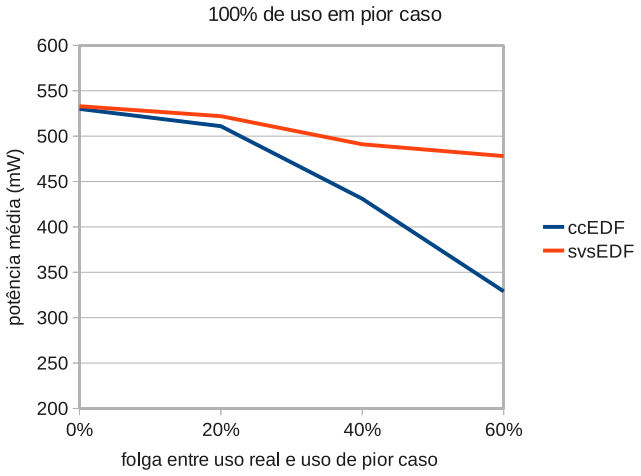

Figure 7. Potência média com utilização real diferente da de pior caso.

varia entre 20% e 100%, ou seja, quando as tarefas não utilizam exatamente o WCET estimado. Através dessa análise é possível verificar a capacidade de cada heurística para se adaptar quando o caso real não é o pior-caso. O que ocorre neste experimento é que, durante a execução, o sistema gera folgas (*slack time*). Como esperado, a heurística dinâmica promove um menor consumo de energia, já que se beneficia das folgas geradas.

VII. Conclusões

A maior contribuição deste trabalho encontra-se na implementação de um ambiente RT-DVFS para o EPOS. Grande parte das implementações de ambientes RT-DVFS utilizam o sistema operacional Linux. O EPOS é um sistema operacional nativamente desenvolvido para a produção de sistemas embarcados, além de seguir a metodologia ADESD [6]. Isto gera um cenário diferenciado das outras implementações

[2] $P \approx a \cdot C \cdot f \cdot V^2$

[3] $E = \int Pdt$: energia é a integral da potência no tempo.

presentes na literatura sobre o tema.

Ainda, as novas abstrações que foram criadas no sistema permitem facilmente a criação de novas heurísticas. A classe `RTDVFS_Thread` é capaz de fornecer metadados referentes a execução de tarefas periódicas. Utilizando-a em conjunto com os eventos captados por `RTDVFS_Heuristic`, basta estender esta classe para que uma nova política RT-DVFS seja adicionada ao sistema. Vale salientar que as heurísticas continuam fracamente acopladas ao escalonador de tempo real do sistema operacional embarcado, como proposto na criação das mesmas por Pillai [1].

AGRADECIMENTOS

Para execução deste trabalho, Gustavo Roberto Nardon Meira foi financiado pelo programa PIBIC 2010/2011 de iniciação científica do CNPq/UFSC; Arliones Hoeller Jr. foi financiado em parte pela CAPES, projeto RHTVD 006/2008.

REFERENCES

[1] P. Pillai and K. G. Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *Proceedings of the eighteenth ACM symposium on Operating systems principles*, ser. SOSP '01. New York, NY, USA: ACM, 2001, pp. 89–102. [Online]. Available: http://doi.acm.org/10.1145/502034.502044

[2] J. D. Lin, W. Song, and A. M. Cheng, "Real-energy: a new framework and a case study to evaluate power-aware real-time scheduling algorithms," in *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, ser. ISLPED '10. New York, NY, USA: ACM, 2010, pp. 153–158. [Online]. Available: http://doi.acm.org/10.1145/1840845.1840877

[3] D. Snowdown, S. Ruocco, and G. Heiser, "Power management and dynamic voltage scaling," in *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, New Jersey, USA, set. 2005.

[4] Y. Zhu and F. Mueller, "Exploiting synchronous and asynchronous dvs for feedback edf scheduling on an embedded platform," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 3:1–3:26, December 2007. [Online]. Available: http://doi.acm.org/10.1145/1324969.1324972

[5] H. Marcondes, R. Cancian, M. Stemmer, and A. A. Fröhlich, "On design of flexible real time schedulers for embedded systems," in *International Symposium on Embedded and Pervasive Systems*, Vancouver, Canada, ago. 2009, pp. 382–387.

[6] A. A. Fröhlich, *Application-Oriented Operating Systems*. Sankt Augustin: GMD - Forschungszentrum Informationstechnik, 2001. [Online]. Available: http://www.lisha.ufsc.br/pub/aoos.pdf

[7] Intel, *Intel PXA255 Processor: Developer's Manual*, Intel, jan. 2004.

[8] J.-J. Chen and C.-F. Kuo, "Energy-efficient scheduling for real-time systems on dynamic voltage scaling (dvs) platforms," in *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, ser. RTCSA '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 28–38. [Online]. Available: http://dx.doi.org/10.1109/RTCSA.2007.37

[9] F. V. Polpeta and A. A. Fröhlich, "Hardware Mediators: a Portability Artifact for Component-Based Systems," in *International Conference on Embedded and Ubiquitous Computing*, ser. Lecture Notes in Computer Science, vol. 3207. Aizu, Japan: Springer, Aug. 2004, pp. 271–280.