DVFS de tempo-real para o sistema operacional EPOS

Gustavo Roberto Nardon Meira e Antônio Augusto Fröhlich
Laboratório de Integração Software/Hardware
Universidade Federal de Santa Catarina, UFSC - Brasil
Email: {meira,guto}@lisha.ufsc.br

Arliones Hoeller Jr.
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina, UFSC - Brasil
Email: arliones@das.ufsc.br

***Resumo*—Várias implementações de sistemas RT-DVFS (*Real-Time Dynamic Voltage and Frequency Scaling*) vem sendo estudadas na última década. Em sua maioria, estas implementações são realizadas em sistemas Linux, que exigem modificações para suporte a tempo real. Este trabalho apresenta o projeto e implementação do suporte a DVFS para os escalonadores de tempo-real do EPOS (Embedded Parallel Operating System), um sistema operacional de tempo real orientado a aplicação. O projeto dá suporte à inserção de heurísticas RT-DVFS de forma fracamente acoplada ao escalonador do sistema. Para avaliar o projeto, as heurísticas RT-DVFS propostas por Pillai [1] foram implementadas. Os experimentos conduzidos demonstraram o ganho energético de heurísticas utilizadas em conjunto com o escalonador EDF (Earliest Deadline First) do EPOS, validando a implementação.**

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 EPOSé um sistema operacional que segue a metologia ADESD (*Application-Driven Embedded System Design*) [6], estratificando ainda mais o cenário utilizado neste trabalho.

Neste trabalho, utiliza-se o porte do EPOSpara 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. A seção II o ambiente experimental utilizado no trabalho. A seção III dá uma breve introdução ao sistema EPOSe mostra o funcionamento de seu ambiente de tempo real. Em sequência, a seção IV descreve o projeto e implementação do da extensão RT-DVFS do EPOS. A seção V mostra os resultados dos experimentos conduzidos. Finalmente, a seção VI 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.

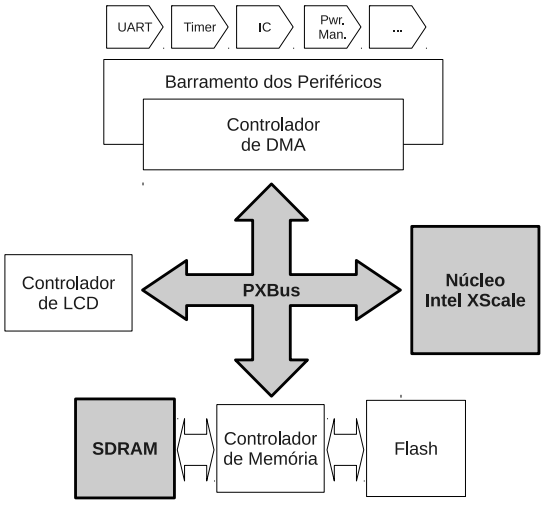

Figura 1. Topologia de componentes do PXA255.

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.

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

Tabela I
SUBCONJUNTO DE CONFIGURAÇÕES DO PXA255 UTILIZADAS NESTE TRABALHO.

III. ESCALONAMENTO DE TEMPO REAL NO EPOS

O EPOS(*Embedded Parallel Operating System*) [8] é um sistema operacional dirigido a aplicações embarcadas de alto desempenho. 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 EPOSé 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, na Figura 2, como sendo o atributo `period` da classe `Alarm`. A rotina associada à tarefa periódica é representada na mesma imagem pelo atributo `entry`, herdado por `Periodic_Thread` da classe `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` 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 um objeto *functor* (instância da classe `Handler` apresentada na Figura 2), 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.

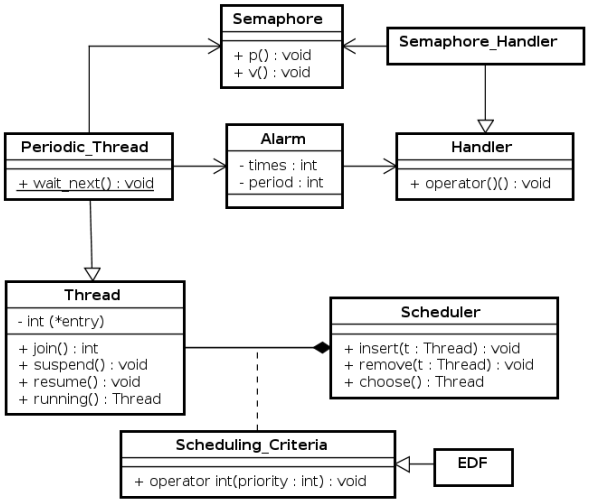

Figura 2. Diagrama de classes com a abstração `Periodic_Thread`

Ao retomar a execução, ela volta ao ponto onde invocou `Periodic_Thread::wait_next`, obedecendo ao comportamento do laço escrito na aplicação.

IV. SOLUÇÃO RT-DVFS PARA O EPOS

Para implementar as heurísticas RT-DVFS, o escalonador precisar 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 3.

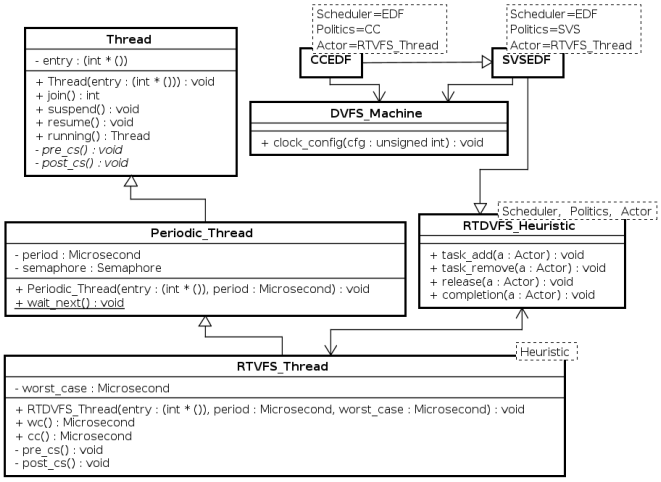

Figura 3. Diagrama de classes da solução `RTVFS_Thread` do EPOS

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 4 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.

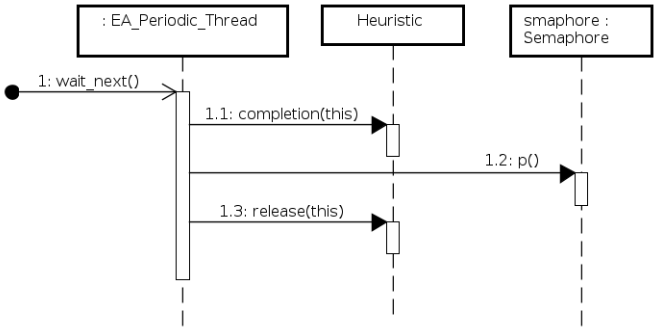

Figura 4. Diagrama de sequência demonstrando o comportamento do método `wait_next` da classe `RTDVFS_Thread`.

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 instâcia 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;

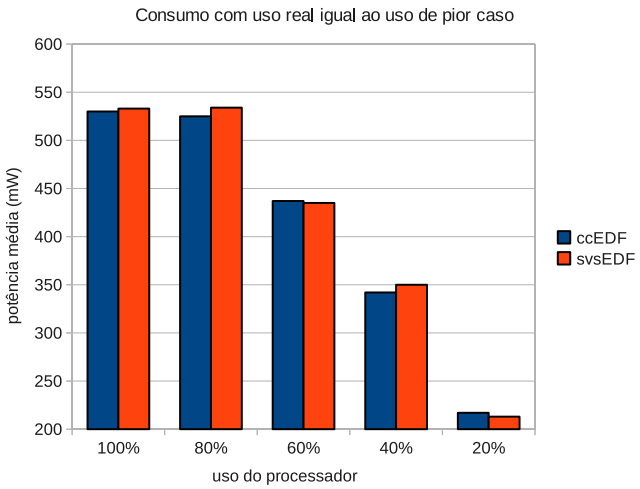

Figura 5. Potência média com utilização real igual à de pior caso.

- `Politics`: a política a ser utilizada: *Static Voltage Scaling* ou *Cycle-conserving*);
- `Actor`: a abstração que utiliza a heurística, neste trabalho `RTDVFS_Thread`.

V. 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. Como métrica para avaliação, utilizou-se a potência média de todo o sistema. 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 5 mostra a potência média obtida para uma utilização real igual à utilização de pior caso. Neste caso a utilização real de cada tarefa é igual à utilização referente ao pior tempo de computação passado como parâmetro para a construção do objeto `RTDVFS_Thread`. 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 6 mostra o comportamento das heurísticas quando a utilização real das tarefas do sistema varia entre 20% e 100%. Isto significa que, mesmo quando as tarefas tem como parâmetro um pior tempo de computação que causa 100% de utilização, na realidade as tarefas canônicas correspondentes realizam apenas uma taxa fixa entre 20% e 100% do pior caso de computação informado. Como esperado,

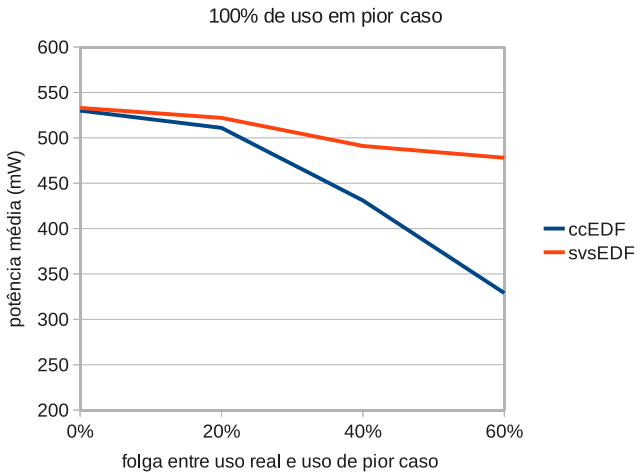

Figura 6. Potência média com utilização real diferente da de pior caso.

a heurística dinâmica promove um menor consumo de energia, já que se beneficia de dados capturados dinamicamente.

VI. 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 EPOSé 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 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.

Referências

[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] T. R. Mück, M. Gernoth, W. Schröder-Preikschat, and A. A. Fröhlich, "Implementing OS Components in Hardware using AOP," *SIGOPS Operating Systems Review*, vol. 46, no. 1, pp. 64–72, 2012. [Online]. Available: http://dx.doi.org/10.1145/2146382.2146395

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

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

[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.