Exploração de Desempenho, Consumo Dinâmico e Eficiência Energética em MPSoCs

Liana Duenha[1,2], Guilherme Madalozzo[3], Thiago Santiago[1], Fernando Gehm Moraes[3], Rodolfo Azevedo[1]

[1]Universidade Estadual de Campinas (IC-UNICAMP). Campinas, Brasil. *rodolfo@ic.unicamp.br, thiago.santiago@lsc.ic.unicamp.br*

[2]Faculdade de Computação – Universidade Federal de Mato Grosso do Sul (FACOM–UFMS) Campo Grande, Brasil. *lianaduenha@facom.ufms.br*

[3]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS). Porto Alegre, Brasil. *guilherme.madalozzo@acad.pucrs.br, fernando.moraes@pucrs.br*

Abstract. *To explore performance and energy efficiency in MPSoCs designs, it is necessary a system level simulation infrastructure with resources for evaluating energy consumption in the early stages of the design. This paper presents an extension of a framework for MPSoCs designs to support dynamic voltage and frequency scaling (DVFS) in MPSoCs simulators and evaluates three DVFS mechanisms. The experiments show that applying DVFS can save considerably the energy consumption, with negligible loss of performance.*

Resumo. *Para realizar exploração de desempenho e eficiência energética em projetos de MPSoCs, faz-se necessária uma infraestrutura de simulação em nível de sistema que forneça recursos para avaliar consumo de energia nos estágios iniciais do projeto. Este artigo apresenta uma extensão de um framework para projetos MPSoCs que fornece suporte à escalabilidade dinâmica de voltagem e frequência (DVFS) em simuladores de MPSoCs e a avalia três mecanismos de DVFS. Os experimentos mostram que a aplicação de técnicas de DVFS pode economizar energia consideravelmente, com insignificante perda de desempenho da simulação.*

1. Introdução

Estimativas sobre consumo de energia de um sistema dependem intrinsecamente do projeto final do *chip*. Embora precisa, esta metodologia demanda longos períodos de síntese e de simulação em nível de abstração muito próximo ao hardware, tornando-a inviável para avaliação de eficiência energética nos estágios iniciais de um projeto de sistemas multiprocessados. Incluir suporte à avaliação de consumo de potência e energia em simuladores de desempenho descritos em nível de sistema melhora a exploração de espaço de projeto e permite avaliar o compromisso existente entre desempenho e consumo antecipadamente [Spiliopoulos et al. 2013].

Inicialmente propostas em 1994 por Weiser et al. [Weiser et al. 1994], técnicas de escalabilidade dinâmica de voltagem e frequência ou DVFS (do inglês, *Dynamic Voltage and Frequency Scaling*) mostraram-se altamente eficazes para diminuir consumo de potência em sistemas computacionais. Uma evolução natural dos simuladores de sistema

é a inclusão de suporte à DVFS, tornando possível a rápida exploração de mecanismos de controle de frequência em tempo de projeto, otimizando o perfil de consumo do sistema.

O objetivo deste artigo é apresentar modelos de consumo de potência de componentes de hardware e validar o suporte a técnicas de DVFS em simuladores de sistemas *multicore* disponíveis no MPSoCBench [Duenha et al. 2014]. Mais especificamente, as principais contribuições do artigo são as seguintes:

- Suporte dinâmico à mudança de frequência e voltagem, consequentemente, perfilamento de consumo de energia nos processadores do MPSoCBench;
- Estimativa de consumo para as caches de dados, cache de instruções e integração do simulador DRAMSim2 [Rosenfeld et al. 2011] ao controlador de memória do MPSoCBench;
- Integração de um modelo de consumo às interconexões baseadas em NoCs obtido por meio de caracterização de consumo de uma NoC RTL;
- Avaliação de três técnicas de DVFS: baseada em software, baseada em taxa de uso de CPU e baseada em métricas de energia. Essa última caracteriza uma importante contribuição do trabalho, pois não está implementada em nenhum dos simuladores pesquisados;
- Demonstração de que estas extensões podem ser utilizadas para estudos relativos ao consumo de plataformas multiprocessadas, sem comprometer significantemente o desempenho dos simuladores.

O restante deste artigo está organizado da seguinte forma: A Seção 2 revisa os principais trabalhos sobre ferramentas e técnicas de DVFS; A Seção 3 apresenta a infraestrutura de simulação e componentes dos MPSoCs utilizados; A Seção 4 descreve a aplicação de três técnicas de DVFS, a Seção 5 avalia tais técnicas e a Seção 6 conclui o artigo.

2. Trabalhos Relacionados

Esta Seção descreve os principais simuladores com suporte à DVFS e alguns mecanismos de DVFS já avaliados na literatura.

Simuladores DVFS: Embora existam vários simuladores de desempenho, poucos deles possuem recursos para avaliação de consumo de potência ou energia. Wattch [Brooks et al. 2000] forneceu uma das primeiras ferramentas para lidar com questões relativas ao consumo, e foi largamente utilizada como extensão para diversos simuladores de desempenho, por exemplo, o simulador SimpleScalar. McPAT [Li et al. 2009] é um *framework* de estimativa de consumo e área também muito utilizado como extensão em simuladores de desempenho. Ao invés de produzir dados de consumo baseado em simulação, esta ferramenta possui recursos estatísticos para a estimativa de tais valores que, embora precisos, não refletem o comportamento dinâmico do sistema. O Simulador Gem5 [Binkert et al. 2011] foi combinado com o McPAT para prover estimativas de consumo do sistema; ele coleta estatísticas de desempenho oferecidas pelo simulador e os utiliza como dados de entrada para a ferramenta McPAT para forncer estimativas finais *offline*. Frequência e voltagem são parâmetros esperados pelo McPAT; assim, não é permitido que o projetista modifique estes valores a menos que reinicie o projeto do sistema, e isso é uma importante limitação da ferramenta. Para minimizar estas limitações, Spiliopoulos et al. [Spiliopoulos et al. 2013] propuseram transformar Gem5

em um simulador para estudos de DVFS, incluindo declaração online de frequência e voltagem, um controlador de DVFS e bibliotecas para suporte a DVFS. A limitação desta infraestrutura é que não há modelagem de estado de inatividade ou sensores de monitoramento de potência e temperatura.

Abordagens de DVFS: Kong et al.[Kong et al. 2008] propuseram uma técnica de gerenciamento de consumo de energia, a qual usa um algoritmo de DVFS para gerenciar adaptativamente o consumo baseado em perfilamentos realizados previamente. É calculado o valor de EDP (do inglês, *"energy delay product"*) de cada aplicação, com o objetivo de caracterizar o melhor estado de potência para cada uma delas. A métrica EDP é baseada em consumo de energia (em Joules), CPI (ciclos de *clock* por instrução) e tempo de ciclo de *clock*. Todas as informações são armazenadas em tabelas que são consultadas durante a execução, para que seja feita a escolha mais adequada de frequência e voltagem. Yadav et al. [Yadav et al. 2013] apresentaram LAURA-NoC, uma abordagem de rede em *chip* para exploração de técnicas de DVFS, na qual controladores locais determinam automaticamente a frequência e a voltagem mais adequadas, eliminando a necessidade de um controlador global.

3. Modelos de Sistemas do MPSoCBench

MPSoCBench [Duenha et al. 2014] é uma infraestrutura de simulação de MPSoCs adequada para desenvolvimento e avaliação de novas ferramentas, medologias, aplicações e componentes de hardware de sistemas multiprocessados descritos em alto nível de abstração. Esse conjunto de ferramentas é de código aberto, fácil instrumentação e permite a construção e simulação rápida de centenas de plataformas multicore.

São disponibilizados quatro modelos de processadores (ARM, MIPS, SPARC, and PowerPC) em modelos de plataformas multicore com 1, 2, 4, 8, 16, 32 e 64 *cores*, diferentes dispositivos de interconexão, memória compartilhada, caches com diferentes políticas de escrita e substituição, controlador de interrupções, dispositivo para controle de concorrência e outros IPs, todos conectados por interfaces ou sockets de interconexão TLM2. Os processadores são baseados na ADL ArchC [Azevedo et al. 2005], uma Linguagem de Descrição de Arquiteturas baseada em SystemC.

Um *script* de configuração gera o código apropriado para os processadores e demais componentes de acordo com parâmetros fornecidos pelo usuário. Os modelos são compilados e link-editados para criar um simulador MPSoC. As aplicações solicitadas pelo usuário são compiladas para a arquitetura alvo utilizando o cross-compilador correspondente, e o código executável das aplicações é armazenados em diretórios de execução, juntamente com os executáveis do MPSoC para simulação futura. Após a simulação, os resultados são verificados e é gerado um relatório com estatísticas. A Figura 1 ilustra o processo de criação e simulação de um MPSoC utilizando o MPSoCBench.

3.1. Modelos de Consumo de Potência dos Processadores e Cache

O MPSoCBench contém originalmente modelos de consumo de potência para os processadores SPARC e MIPS utilizando ferramentas de estimativas descritas em [Guedes et al. 2013]. Tais modelos foram elaborados a partir de descrições RTL de código aberto dos processadores LEON e PLASMA, que são compatíveis com as arquiteturas do SPARC e MIPS, respectivamente. O processo de caracterização de consumo é

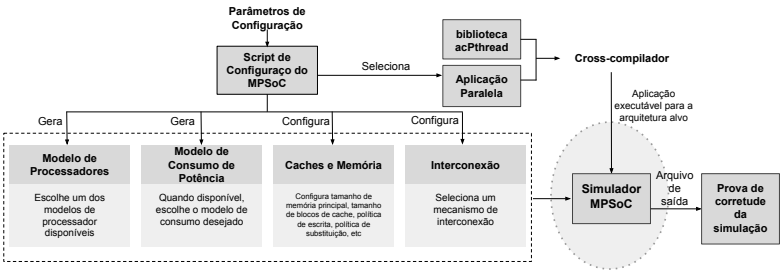


Figura 1. Processo de criação e simulação no MPSoCBench

baseado na metodologia proposta por Tiwari [Tiwari et al. 1994] e está descrito em maiores detalhes em [Guedes et al. 2013].

Como primeira contribuição deste trabalho, adicionamos ao MPSoCBench um modelo de consumo das memórias cache a partir da caracterização de 3 estados possíveis de energia: leitura, escrita e inativo. A energia de leitura (E_{read}), energia de escrita (E_{write}), potência de inatividade (P_{idle}) e o tempo de acesso (T_{access}) ficam armazenados em um arquivo que identifica cada cache. Os 4 valores foram obtidos pela interface web da ferramenta CACTI [Muralimanohar et al. 2009]. A cada acesso de leitura ou escrita, a energia consumida pela operação e o tempo de acesso são incrementados em variáveis. A energia gasta pela inatividade é dada pela multiplicação da potência de inatividade com o valor de tempo de inatividade. Somando a energia consumida com a energia inativa e dividindo pelo tempo de simulação, dá-se a potência média da cache.

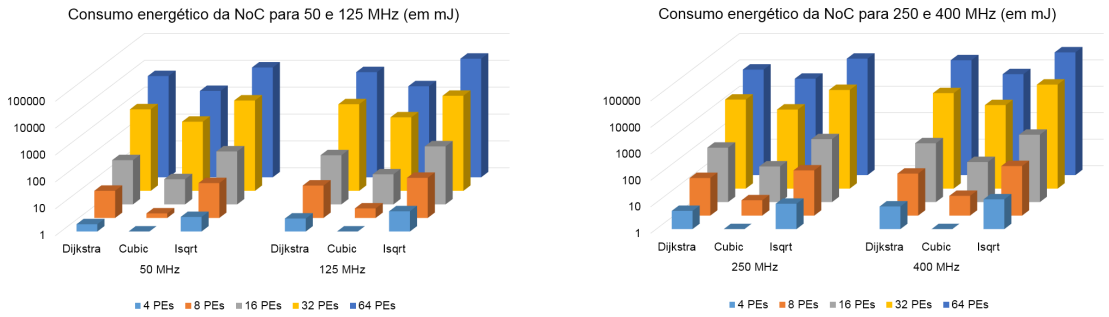
$$P_{media} = \sum(E_{read} + E_{write} + (T_{total} - T_{access}) * P_{idle})/T_{total} \quad (1)$$

Como não existe um modelo de consumo para a memória principal do MPSoCBench, realizamos a sua integração como DRAMSim2, um simulador de memória RAM com precisão de ciclos que contém módulos para gerenciar o sistema de armazenamento e barramentos internos de comunicação entre eles [Rosenfeld et al. 2011].

3.2. Modelo de Consumo da NoC

Martins et. al [Martins et al. 2014] mostram que a dissipação de potência de cada roteador da rede ocorre em função da taxa de recepção dos seus *buffers*. Consequentemente, a caracterização de consumo de um roteador tem como principal parâmetro a quantidade de portas de entrada e saída. O processo de caracterização utilizou um modelo de NoC RTL sobre a tecnologia PDK45. Para cada roteador, o consumo de energia é calculado sobre a quantidade de ciclos inativos e ciclos para transmissão no roteador, considerando que roteadores da rede podem possuir diferente quantidade de portas, dependendo da sua localização. Os roteadores do MPSoCBench utilizamos o mesmo algoritmo de roteamento e políticas de abitragem da NoC RTL utilizada na caracterização.

As Figuras 2(a) e 2(b) mostram os resultados de consumo de energia na NoC com 4, 8, 16, 32 e 64 *cores* executando três benchmarks. Todas as simulações foram executadas sobre frequências 50, 125, 250 e 400MHz. O eixo y representa o consumo da NoC em mJ em escala logaritmica. O eixo x contém os benchmarks executados e as correspondentes frequências aplicadas.



(a) (b)

Figura 2. Resultados de consumo em mJ em NoCs a 50, 125, 250 e 400 Mhz.

3.3. Aplicações e Biblioteca *acPthread*

O MPSoCBench inclui uma biblioteca de emulação de funcionalidades da biblioteca POSIX Threads (PThreads), a qual foi denominada *acPthread*. Esta biblioteca habilita o gerenciamento de *threads*, implementação de barreiras, recursos para exclusão mútua, semáforos e variáveis condicionais.

4. DVFS no MPSoCBench

Técnicas de DVFS são comumente implementadas por definição de estados de frequência e de voltagem e algoritmos para escolha dos estados mais adequados durante a execução das aplicações. As tabelas de caracterização de potência utilizadas neste artigo contém informações relativas á caracterização variando o par frequência e voltagem. Entretanto, os experimentos realizados neste trabalho levaram em consideração apenas variações na frequência.

As subseções seguintes descrevem a infraestrutura básica para implementação de DVFS e apresentam três técnicas de DVFS que a utilizam para alcançar economia de energia.

4.1. Infraestrutura para aplicação de DVFS

Com relação aos componentes de hardware, introduzimos um dispositivo global à rede chamado DVFS-IP para gerenciar a frequência em que cada *core* opera, e controladores locais de DVFS (um para cada *core*) para gerenciar a frequência localmente. A Figura 3 mostra a arquitetura geral do sistema, incluindo os controladores locais e o global DVFS-IP.

O DVFS-IP é inicializado com as frequências disponíveis para todos os *cores* do sistema e pode gerenciar a frequência globalmente. Esse dispositivo está no espaço de endereçamento do sistema e permite que a aplicação que executa em um *core* possa modificar a frequência de outro *core* do sistema. Para implementar esse mecanismo, o DVFS-IP envia mensagens para os controladores locais de DVFS para que modifiquem a frequência do *core* em que estão conectados.

Cada controlador local contém informações sobre as frequências disponíveis para o *core* em que está conectado, e é responsável por realizar as atividades relativas à troca de frequência. Uma vez que um processador modifica sua frequência, novas informações de consumo de potência são utilizadas para estimativa de energia consumida no *core*.

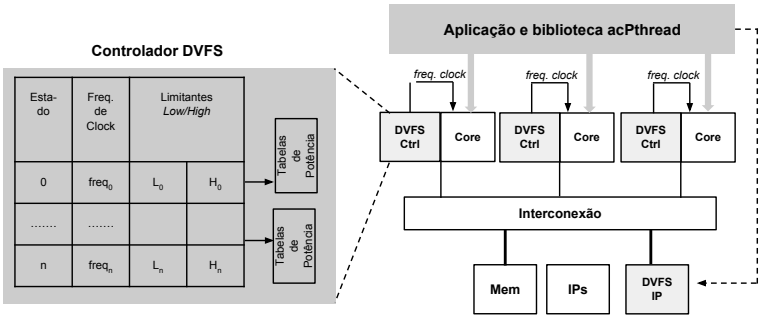


Figura 3. Arquitetura geral do sistema com suporte à DVFS

4.2. DVFS controlado por Software: DVFS-SW

Para implementar DVFS gerenciado por software, estendemos a biblioteca *acPthread* para permitir a mudança de frequência baseado na carga de trabalho das aplicações. As aplicações são baseadas em paralelismo cooperativo sobre uma memória global compartilhada, com controle de concorrência implementado por funcionalidades da biblioteca *acPthread*.

Atuamos, assim, sobre as funcionalidades de criação e pontos de sincronização entre *threads* de acordo com o seguinte fluxo: um dos processadores atua como mestre e envia tarefas para que os outros *cores* denominados escravos. Enquanto o mestre está lendo arquivos de entrada e preparando argumentos, os demais estão em aguardando em uma barreira. Assim, o mestre mantém os demais *cores* em um estado de baixo consumo até que as tarefas estejam prontas para serem executadas. Por meio de recursos da biblioteca acPthread, os *cores* escravos são notificados para que modifiquem sua frequência logo antes de iniciarem a execução de uma *thread* da aplicação.

A Tabela 1 mostra o resultado da potência média por instrução coletada a partir de simuladores de plataformas MIPS *multicore* executando as 17 aplicações paralelas do MPSoCBench, com e sem o gerenciamento de frequência descrito acima e utilizando a maior quantidade de cores possível para executar cada aplicação. Os mecanismos de DVFS escolhem entre duas frequências: 50Mhz e 400Mhz. O processador mestre inicia a 400Mhz e os processadores escravos iniciam e mantêm-se em 50Mhz enquanto estão aguarando por tarefas. Imediamentamente antes de iniciar a execução de uma *thread*, os processadores escravos são notificados para que elevem a sua frequência para 400Mhz. A medida que terminam a execução das *threads*, os *cores* escravos voltam a operar a 50Mhz até que haja outra demanda. Sem DVFS, os *cores* mantêm-se a 400Mhz todo o tempo.

A melhoria no consumo médio é significantemente maior para as aplicações baseadas em paralelismo cooperativo, nas quais é possível tirar vantagem dos diversos pontos de sincronização para economizar energia. No caso das aplicações *Multisoftware* nas quais várias aplicações *single-threaded* executam em cores distintos da plataforma com praticamente nenhuma sincronização entre elas, a economia de energia mínimo.

4.3. DVFS com base em energia: DVFS-ES

Apresentamos um mecanismo de auto-seleção de frequência com base na métrica de EDP (do inglês, “energy delay product”), na qual frequência mais adequada para um processador é escolhida pelo seu controlador local utilizando como parâmetro a melhor relação

Tabela 1. Ganhos relativos à potência média com aplicação de DFVS-SW

Aplicações	Quantidade de Cores	Ganhos em Potência Média
Basicmath	64	27,3%
Dijkstra	64	73,3%
FFT	16	85,7%
LU	16	81,1%
SHA	64	71,7%
Stringsearch	64	63,7%
Susan-corners	32	64,3 %
Susan-edges	32	6,9%
Susan-smoothing	8	28,4%
Water	16	81,2%
Water-spatial	8	75,1%
Multi-parallel	4	11,3%
Multi-8	8	2,6%
Multi-16	16	1,4%
Multi-network-automotive	4	0,1%
Multi-office-telecomm	4	0,1%
Multi-security	4	0,1%

entre energia e desempenho encontrada até o momento. A ideia geral é realizar um perfilamento nos primeiros estágios na simulação a fim de obter um "selo de energia" ou ES (do inglês, *energy stamp*) que leve em consideração tanto a energia consumida quanto a quantidade de instruções executadas no período (desempenho). Adotamos o modelo descrito abaixo:

- O sistema possui n *cores* e cada *core* possui m estados de potência (que correspondem a tabelas obtidas por prévia caracterização, uma tabela para cada frequência disponível);
- Estabelecemos tempo de troca de frequência (SWITCHING_TIME) de 20 μs. Esse valor corresponde à latência de transição entre duas frequências distintas e foi definido com base em [Spiliopoulos et al. 2013];
- Cada tabela de potência possui o valor de EPI ou energia por instrução (do inglês, *"energy per instruction"*) em nJ para cada instrução da ISA;
- Para cada processador, definimos o conjunto de frequências disponíveis, $F = \{f_0, f_1, ..., f_m\}$;
- Δ_T é o tempo ou período utilizado na fase de avaliação para calcular ES de cada frequência;
- $Instr_{\Delta_T}$ é a quantidade de instruções executadas durante Δ_T unidades de tempo

Fase de Inicialização: Antes de iniciar a simulação, informações de potência por instrução referentes às frequências disponíveis para cada *core* são armazenados em estruturas de dados dos controladores e todos os *cores* são inicializados com uma frequência inicial (*default*);

Fase de Avaliação: Cada simulador de processador executa por Δ_T unidades de tempo (em nanossegundos) à frequência f, e calcula o valor do selo de energia (ES)

como descrito a seguir:

$$ES_f = \frac{(\sum_{\Delta_T} EPI)}{Instr_{\Delta_T}} \quad (2)$$

$\sum_{\Delta_T} EPI$ é a soma da energia consumida (em Joules) em Δ_T unidades de tempo. ES é definido com base na métrica EDP definida em [III et al. 2012] e largamente utilizada a partir de então. A diferença fundamental é que em [III et al. 2012] as informações de desempenho são estimadas e na metodologia que estamos apresentando, as informações de desempenho são coletadas durante a simulação. No final da fase de avaliação, o controlador DVFS de cada *core* seleciona a frequência f a partir do valor mínimo de ES encontrado, como descrito a seguir:

$$ES_f = min\{ES_j\}, 1 \leq j \leq m \quad (3)$$

Fase de Simulação: O perfil de consumo de energia de cada *core* sofre alterações durante a simulação, dependendo das instruções executadas e dos atrasos para acesso à memória ou I/O. Portanto, a escolha de Δ_T é significante para tornar mais eficiente esta técnica. A maioria das aplicações do MPSoCBench possui centenas de milhões de instruções e, assim, podemos explorar uma significante parte da aplicação para fazer a melhor escolha de frequência por *core*. Durante a fase de simulação, os controladores de DVFS monitoram o valor corrente de ES (ES_{cur} a cada Δ_T unidades de tempo, e verificam se existe outra frequência j na qual $ES_j < ES_{cur}$ e, nesse caso, trocam a frequência do processador novamente.

4.4. DVFS com base em taxa de uso de CPU: DVFS-CPU

Apresentamos também um mecanismo de DVFS para explorar dinamicamente o consumo de energia por *core* com base na taxa de uso de CPU. Este mecanismo utiliza os seguintes parâmetros, além dos que foram definidos anteriormente:

- O sistema possui n *cores* e cada *core* possui m estados de potência (que correspondem a tabelas obtidas por prévia caracterização, uma tabela para cada frequência disponível);
- Para cada frequência f_j ($0 \leq j < m$) disponível, definimos:
  - L_j: limitante inferior para a taxa de uso de CPU, e
  - H_j: limitante superior para a taxa de uso de CPU;
- max_rate: taxa máxima de uso de CPU em Δ_T unidades de tempo;
- min_rate: taxa mínima de uso de CPU em Δ_T unidades de tempo;

A fase de inicialização deste mecanismo é igual a do seu antecessor (DVFS-ES), incluindo a inicialização dos limitantes inferior e superior para cada frequência disponível. A métrica que dá suporte a este mecanismo é cpu_rate, ou taxa de uso de CPU, calculado durante a execução. Para cada Δ_T unidades de tempo, o controlador DVFS calcula a taxa de uso de CPU baseado no tempo de espera para acesso à memória, I/O ou outros IPs (cpu_wait_time) e o tempo de simulação corrente. O tempo de espera é facilmente obtido pela diferença entre o tempo de início de uma requisição (t_{req}) e o seu

tempo de resposta (t_{resp}). Esta métrica é fortemente relacionada com a métrica CPI (ciclos por instrução) do processador. Para cada requisição r feita pelo processador à rede, nos primeiros Δ_T unidades de tempo, definimos:

$$cpu\_wait\_time_{\Delta_T} = \sum_{w}(t\_resp_w - t\_req_w), \tag{4}$$

Portanto, a taxa de uso de CPU ($cpu\_rate_{\Delta_T}$) é obtida a partir de Δ_T e de cpu_wait_time:

$$cpu\_rate_{\Delta_T} = \frac{\Delta_T - cpu\_wait\_time_{\Delta_T}}{\Delta_T} \tag{5}$$

Neste mecanismo não há fase de avaliação. Durante a fase de simulação, a cada $N * \Delta_T$ (N também configurável) unidades de tempo, o controlador DVFS usa o valor de cpu_rate, max_rate e min_rate para melhorar a estimativa de limitante inferior e limitante superior para cada frequência j, e verifica se é necessário realizar trocas de frequência, comparando a taxa de CPU do core com os limitantes previamente definidos. O mecanismo de seleção é descrito a seguir:

- Considere que o *core* executa à frequência j;
- Se cpu_rate está entre L_j e H_j, não troque a frequência;
- Caso contrário, encontre a frequência f tal que o intervalo entre seus limitantes inferior e superior (L_f e H_f) contém a corrente taxa de uso de CPU (cpu_rate);

5. Avaliação das Técnicas Propostas

Escolhemos quatro aplicações paralelas (Dijkstra, Basicmath, SHA e Stringsearch) e um simulador de sistema paralelo com 16 MIPS conectados em rede do tipo *mesh*. Cada aplicação possui 4 *threads* e cada *thread* executa em um *core*. Assim, os 16 *cores* são divididos em 4 *clusters* com 4 *cores* cada.

Inicialmente, aplicamos a técnica de DVFS-ES utilizando quatro diferentes frequências (50Mhz, 125Mhz, 250Mhz e 400Mhz). A caracterização de energia por instrução foi realizada sobre a ferramenta FreePDK de 45nm, utilizando as mesmas frequências. Simulamos a execução das aplicações na plataforma utilizando diferentes frequências com e sem o mecanismo de DVFS. Os dados de DVFS-ES referem-se à aplicação da técnica de DVFS baseada em selo de energia; DVFS-CPU refere-se à técnica de DVFS baseado em taxa de uso de CPU; DVFS-SW é a técnica de DVFS controlado por software.

A notação MX/Y/Z/W na primeira coluna significa que inicializamos manualmente (M) os *cores* utilizando as frequências X,Y,Z e W da forma mais balanceada possível. Por exemplo, a notação M50/125/250/400 refere-se à plataforma *16-cores* onde 4 *cores* foram configurados a 50Mhz, 4 *cores* a 125Mhz, 250 *cores* a 250Mhz e 4 *cores* a 400 Mhz.

Para calcular a melhoria em consumo de energia, medimos a energia total consumida pelos *cores* utilizando os mecanismos de DVFS e comparamos esse valor com o consumo obtido em 15 diferentes simulações sem DVFS. Escolhemos arbitrariamente o

valor de $\Delta_T = 250\mu s$ para todas as simulações e calibramos DVFS-CPU para reavaliar os limitantes inferiores e superiores a cada 100x$\Delta_T\mu s$. Utilizamos o seguinte mecanismo para estabelecer os limitantes para cada uma das possíveis frequências:

Tabela 2. Mecanismo para estabelecimento dos limitantes para cada frequência disponível

L_{50}	0	L_{125}	$H_{50} - H_{50} * 25\%$
H_{50}	$min\_rate/2$	H_{125}	min_rate
L_{250}	$H_{125} - H_{125} * 25\%$	L_{400}	$H_{250} - H_{250} * 25\%$
H_{250}	$(max\_rate + min\_rate)/2$	H_{400}	100

Para evitar sucessivas trocas de frequência de um processador quando o valor de cpu_rate está próximo dos limitantes entre duas frequências, configuramos uma sobreposição de 25% entre o limitante superior de uma frequência e o limitante inferior da frequência seguinte. Essa taxa de sobreposição foi escolhida arbitrariamente. A Tabela 3 mostra que a aplicação de DVFS-ES e DVFS-SW resultam em economia de energia em 14 das 15 configurações manuais utilizadas, e DVFS-CPU apresentou melhoria no consumo em todas as configurações.

Tabela 3. Economia de energia utilizando DVFS

Configuração Manual de Frequência	**Ganhos**		
	(DVFS-ES)	**(DVFS-CPU)**	**(DVFS-SW)**
M125	-1,98%	6,17%	-0,20%%
M50/250/400	2,61%	10,40%	4,31%
M50/M125	5,66%	13,20%	7,30%
M400	5,84%	13,36%	7,48%
M250/400	6,79%	14,24%	8,42%
M125/400	8,15%	15,49%	9,75%
M125/250/400	8,58%	15,89%	10,17%
M250	8,64%	15,95%	10,24%
M50/125/250/400	9,03%	16,30%	10,62%
M125/250	9,38%	16,63%	10,96%
M50/125/400	12,85%	19,81%	14,37%
M50	13,53%	20,44%	15,04%
M50/400	13,71%	20,61%	15,21%
M50/250	14,12%	20,99%	15,62%
M50/125/250	14,40%	21,25%	15,92%

A Figura 4 mostra a Energia (em Joules) e Tempo (em segundos) normalizado pela energia e tempo obtidos pela simulação M125, a configuração com menor consumo dentre todas as configurações manuais. A Figura 5 mostra o valor de EDP para simulações com e sem DVFS, normalizado pelo EDP da configuração M125. EDP é calculado pela multiplicação da energia (em Joules) pelo tempo de simulação (em segundos).

O impacto no desempenho da simulação com a aplicação dos nossos mecanismos de DVFS foi de 3% para DVFS-ES, 4% para DVFS-CPU e desprezível para DVFS-SW.

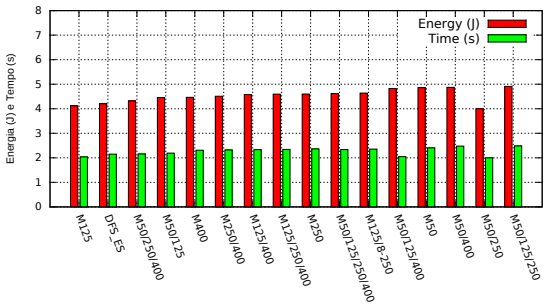


Figura 4. Energia e Tempo de simulações de sistema com 16 *cores*, normalizados pela Energia e Tempo da configuração M125.

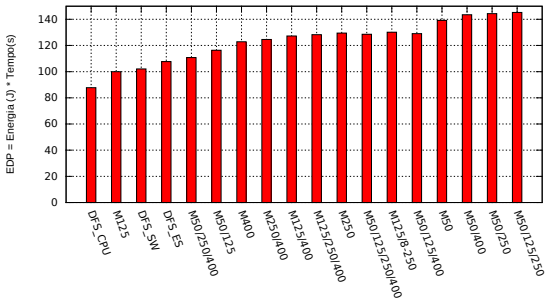


Figura 5. EDP normalizado pelo EDP da configuração M125.

Os ganhos com DVFS tendem a aumentar a medida que aumentamos o valor de Δ_T. Entretanto, quanto maior esse valor, maior será o impacto no desempenho do simulador. Isto ocorre pois a simulação será interrompida mais vezes para garantir os cálculos das métricas e limitantes.

6. Conclusão

Este artigo mostrou a inclusão de modelos de potência para componentes de hardware e suporte à DVFS em simuladores de sistemas multiprocessados em único chip. Avaliamos três mecanismos de DVFS e mostramos que estas extensões podem ser utilizadas para avaliar consumo de energia logo nos primeiros estágios do projeto, com perda de desempenho máximo de 4%. A abordagem de DVFS controlada por software (DVFS-SW) e o mecanismo de auto-seleção baseado em energia (DVFS-ES) economizaram energia em 14 das 15 configurações avaliadas. O mecanismo de auto-seleção basesado em taxa de uso de CPU (DVFS-CPU) economizou enengia nas 15 configurações avaliadas.

Referências

Azevedo, R., Rigo, S., Bartholomeu, M., Araujo, G., Araujo, C., and Barros, E. (2005). The ArchC Architecture Description Language and Tools. In *International Journal of Parallel Programming. Vol. 33, No. 5*, pages 453–484.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. (2011). The gem5 simulator. volume 39, pages 1–7.

Brooks, D., Tiwari, V., and Martonosi, M. (2000). Wattch: A framework for architectural-level power analysis and optimizations. *SIGARCH Comput. Archit. News*, 28(2):83–94.

Duenha, L., Guedes, M., Almeida, H., Boy, M., and Azevedo, R. (2014). Mpsocbench: A toolset for mpsoc system level evaluation. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIV)*, INSPEC Number: 14564763, pages 164–171. IEEE.

Guedes, M., Auler, R., Duenha, L., Borin, E., and Azevedo, R. (2013). An automatic energy consumption characterization of processors using archc. *Journal of Systems Architecture*, 59(8):603 – 614.

III, J. H. L., Pedretti, K., Kelly, S. M., Shu, W., abd, K. F., Dyke, J. V., and Vaughan, C. (2012). *Energy-Efficient High Performance Computing: Measurement and Tuning*. Springer New York.

Kong, J., Choi, J., Choi, L., and Chung, S. W. (Nov-2008). Low-cost application-aware dvfs for multi-core architecture. In *Proceedings of 3rd International Conference on Convergence and Hybrid Information Technology*, volume 2 of *ISBN:978-0-7695-3407-7*, pages 106–111, Busan. IEEE.

Li, S., Ahn, J., and Strong, R. (2009). McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures. In *Proceedings of the 42nd Annual International Symposium on Microarchitecture*, pages 469–480.

Martins, A., Silva, D., Castilhos, G., Monteiro, T., and Moraes, F. (2014). A method for noc-based mpsoc energy consumption estimation. In *International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE.

Muralimanohar, N., Balasubramonian, R., and Jouppi, N. P. (2009). Cacti 6.0: A tool to model large caches. *HP Laboratories*, pages 22–31.

Rosenfeld, P., Cooper-Balis, E., and Jacob, B. (2011). Dramsim2: A cycle accurate memory system simulator. In *IEEE-Computer Architecture Letters*, ISSN 1556-6056, pages 16–19, Maryland, USA. IEEE.

Spiliopoulos, V., Bagdia, A., Hansson, A., Aldworth, P., and Kaxiras, S. (2013). Introducing dvfs-management in a full-system simulator. *Proceedings of the IEEE 21st International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS-2013)*, pages 535–545.

Tiwari, V., Malik, S., and Wolfe, A. (1994). Power analysis of embedded software: A first step towards software power minimization. In *IEEE Transactions on VLSI Systems, vol. 2*, page 437–445.

Weiser, M., Welch, B., Demers, A., and Shenker, S. (1994). Scheduling for reduced cpu energy. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA. USENIX Association.

Yadav, M., Casu, M., and Zamboni, M. (Oct, 2013). Laura-noc: Local automatic rate adjustment in network-on-chips with a simple dvfs. In *IEEE Transactions on Symposium on Circuits and Systems II: Express Briefs*, volume 60 of *ISSN:1549-7747*, pages 647 – 651. IEEE.