PampaFreq: Otimizando o EDP de Aplicações Paralelas em Processadores AMD*

Mariana Costa[1], Sandro M. V. N. Marques[1], Thiarles S. Medeiros[1], Fábio D. Rossi[2], Marcelo C. Luizelli[1], Antonio Carlos S. Beck[3] e Arthur F. Lorenzon[1]

[1]Universidade Federal do Pampa – Campus Alegrete – RS – Brazil

[2]Instituto Federal Farroupilha – Campus Alegrete – RS – Brasil

[3]Universidade Federal do Rio Grande do Sul – Porto Alegre – RS – Brasil

mary.costa201025@gmail.com

Abstract. *DVFS (Dynamic Voltage and Frequency Scaling) has been widely used to improve the use of computational resources when running parallel applications. However, parallel applications behave differently and relate in different ways to each DVFS governor. Hence, it is necessary to use optimized methods of DVFS in order to improve the trade-off between performance and energy consumption, represented by EDP (energy-delay product). Given that, through an extensive design space exploration of different DVFS governors, degrees of CPU frequency, and operating mode of boosting techniques with the execution of sixteen parallel applications on three multicore architectures, we propose PampaFreq, a methodology that optimizes EDP on AMD processors considering the characteristics of the application at run time. In the most significant case, PampaFreq optimizes the EDP by 38% when compared to the ondemand DVFS governor.*

Resumo. *O DVFS (Dynamic Voltage and Frequency Scaling) tem sido amplamente utilizado para melhorar o uso dos recursos computacionais quando aplicações paralelas estão sendo executadas. No entanto, as aplicações paralelas têm comportamentos distintos e se relacionam de diferentes maneiras com as políticas de modificação de frequência do DVFS. Neste sentido, é necessário utilizar métodos otimizados de DVFS para melhorar o custo-benefício entre desempenho e consumo de energia, representado pelo EDP (energy-delay product). Dito isso, através de uma extensa exploração de espaço e projeto de diferentes políticas de DVFS, níveis de frequência de operação da CPU e modo de operação de técnicas de boosting com a execução de dezesseis aplicações paralelas em três arquiteturas multicore, nós propomos PampaFreq, uma metodologia que otimiza o EDP em processadores AMD considerando as características da aplicação em tempo de execução. No caso mais significativo, PampaFreq otimiza o EDP em até 38% quando comparado com governor ondemand.*

1. Introdução

A cada nova geração de sistemas computacionais de alto desempenho (HPC - *high-performance computing)*, melhorias significativas são realizadas no *hardware* e *software*

*Este trabalho foi parcialmente financiado pela FAPERGS nos projetos 19/2551-0001224-1 e 19/2551-0001689-1 e PROBIC-FAPERGS

para melhorar o desempenho de aplicações paralelas. No entanto, esta melhoria não pode afetar negativamente a potência consumida pelos componentes de *hardware* (e.g., CPU, memórias *cache* e barramentos de comunicação): enquanto os computadores *exascale* poderão chegar ao consumo de energia correspondente à potência fornecida por uma usina nuclear de médio porte, os processadores de propósito geral têm seu desempenho impactado pelos limites do TDP (*thermal design power*). Portanto, ao executar aplicações paralelas, o objetivo não é apenas melhorar o desempenho, mas também, fazer isso com o menor consumo de energia possível.

Neste sentido, diferentes técnicas têm sido propostas com o objetivo de melhorar o custo benefício entre o desempenho e consumo de energia, representado pela métrica EDP (*energy-delay product*) [Lorenzon and Beck 2019]. Por um lado, considerando que muitas aplicações paralelas não escalam conforme o número de *threads* executando a aplicação aumenta, estratégias que ajustam dinamicamente o número de *threads* conseguem agir no nível de *software* para otimizar o EDP de tais aplicações sem a necessidade de modificação nas características do *hardware* [Lorenzon et al. 2015, Lorenzon et al. 2017]. Por outro lado, dado que componentes de *hardware* permitem que sua frequência e tensão de operação possam ser alteradas em tempo de execução, Sistemas Operacionais implementam técnicas para a gerência de tais componentes [Sartor et al. 2017]. Uma destas técnicas é amplamente conhecida como DVFS.

Um sistema com DVFS ativo tem a habilidade de adaptar a frequência e tensão de operação do processador em tempo de execução de acordo com as características da aplicação que está sendo executada. Com isso, é possível utilizar o período que o processador está em *idle* (e.g., *thread* aguardando dado da memória ou em sincronização) para obter redução cúbica de potência, uma vez que a tensão tem influência quadrática na potência dinâmica do processador. Portanto, quanto maior for a taxa de comunicação/sincronização de uma aplicação paralela, maior será a possibilidade de otimizar o EDP através do ajuste de frequência e tensão de operação do processador. Neste sentido, o DVFS tem se estabilizado como uma técnica promissora para reduzir o consumo de energia de arquiteturas multicore com o mínimo de impacto no desempenho, otimizando assim o EDP da aplicação.

Adicionalmente, técnicas chamadas de "*boosting*" também aplicam DVFS, porém com o objetivo de maximizar o desempenho do processador através do aumento da frequência nominal de operação dentro dos limites do TDP. O princípio de funcionamento da técnica se dá da seguinte maneira: dado um TDP, o processador aumenta a frequência de operação de um grupo de núcleos, enquanto que os núcleos restantes executam a uma frequência menor ou são desligados [Charles et al. 2009]. Estas tecnologias têm sido amplamente utilizadas para melhorar o desempenho de aplicações de diferentes domínios [Charles et al. 2009, Wamhoff et al. , dos Santos Marques et al. 2017, Marques et al. 2019], que contém fases sequenciais e paralelas. Nas fases sequenciais (e.g., seções críticas), o processador aplica o *boosting* na frequência deste núcleo enquanto baixa a frequência de operação dos outros que estão esperando. No entanto, enquanto técnicas de *boosting* podem reduzir o tempo de execução de uma aplicação paralela, o consumo de energia pode não seguir o mesmo comportamento, uma vez que esta técnica aumenta a frequência e tensão de operação do processador.

Portanto, neste artigo nós propomos PampaFreq, uma metodologia para otimizar o

EDP de aplicações paralelas através do ajuste dinâmico da frequência do processador e do modo de operação de *boosting*. PampaFreq considera as características da aplicação (e.g., instruções por ciclo –IPC– e comportamento da memória compartilhada) em tempo de execução para tomar a decisão da frequência ideal e da ativação/desativação das técnicas de *boosting*. Para mostrar sua eficácia, realizamos uma exploração de espaço e projeto com diferentes *governors* do DVFS, faixas de frequência de operação do processador e configuração de Turbo Boost em três plataformas *multicore* (Intel e AMD). Através da execução de dezesseis aplicações paralelas com distintos comportamentos de acesso a memória e utilização do processador, em resumo, as contribuições deste artigo são:

- Mostrar que, muito embora o uso da tecnologia de *boosting* (Turbo Core e Turbo Boost) propicia melhor desempenho, o consumo de energia aumenta de maneira não proporcional, piorando o EDP.
- Mostrar que, o *governor ondemand* não é suficiente para entregar o melhor custo-benefício entre desempenho e energia nos processadores AMD devido a quantidade de níveis de frequência disponível.
- Dada a limitação do *governor ondemand* nos processadores AMD, PampaFreq é capaz de melhorar o EDP em até 38% quando comparado ao *governor ondemand*.

O restante deste artigo está estruturado como segue. A fundamentação teórica relacionada a DVFS e técnicas de *boosting* são apresentadas na Seção 2. A metodologia utilizada no desenvolvimento dos experimentos é detalhada na Seção 3. Os resultados experimentais são discutidos na Seção 4 onde também descrevemos a necessidade do PampaFreq. Os trabalhos relacionados em conjunto com as nossas contribuições são descritas na Seção 5. Por fim, as conclusões são destacadas na Seção 6.

2. Fundamentação Teórica: DVFS e Turbo Boost

Um sistema DVFS implementa diferentes *governors* para controlar a frequência do processador. Os principais disponíveis nos sistemas operacionais atuais incluem: *powersave*, no qual a frequência é setada para o mínimo possível; *performance*, onde a frequência dos núcleos é configurada para o máximo possível; *ondemand*, onde a frequência de operação dos núcleos é ajustada de acordo com a carga de trabalho, sendo este, o padrão em muitas distribuições Linux; *conservative*, similar ao *ondemand*, porém a mudança da frequência de acordo com a carga de trabalho ocorre de maneira mais conservativa; e *userspace*, no qual o usuário define a frequência de operação de cada núcleo/processador.

Técnicas de *boosting* podem melhorar o desempenho do processador através do aumento da frequência de operação nominal dentro dos limites do TDP [Branover et al. 2012]. Em arquiteturas modernas, a frequência da CPU e o gerenciamento de potência dos componentes de *hardware* é controlado pelo ACPI (*advanced configuration and power interface*) [Haj-Yahya et al. 2018]. Ele define os estados de potência e desempenho (*C-states* e *P-states*: *C-states* são usados para reduzir a potência e, consequentemente, o consumo de energia, através do desligamento de subsistemas; e *P-states* permitem modificar os pontos de operação de frequência e tensão dos núcleos que estão ativos). A Figura 1 apresenta como os estados de potência e desempenho são definidos.

Os *C-states* básicos definidos pelo ACPI são: *C0*, onde a CPU/núcleo está ativo e executando instruções; e de *C1* até *Cn*, onde a CPU/núcleo é configurado para reduzir o consumo de energia através do corte do sinal de *clock* e potência dos núcleos que não estão

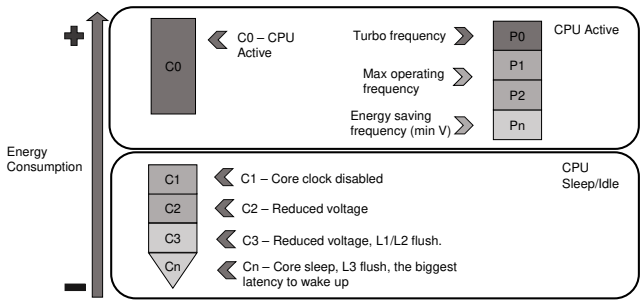


Figura 1. Organização abstrata dos estados de potência e desempenho

sendo utilizados. Nestas situações, quanto mais unidades são desligadas, isto é, quanto maior o valor de *Cn*, menos energia é consumida. Por outro lado, enquanto *C-state* está em *C0*, *P-states* permitem mudar a frequência e tensão de operação das CPUs/núcleos em uso. Em processadores que implementam a tecnologia de *boosting* e usam o padrão ACPI para gerenciamento de potência, o estado P0 representa a frequência de operação mais alta que pode ser alcançada quando o *boosting* está ativo [Haj-Yahya et al. 2018]. Por outro lado, quando está desativado, o estado P1 representa a maior frequência disponível.

3. Metodologia

3.1. Benchmarks

Dezesseis aplicações paralelas de diferentes suítes de benchmarks já paralelizadas com OpenMP foram utilizadas.

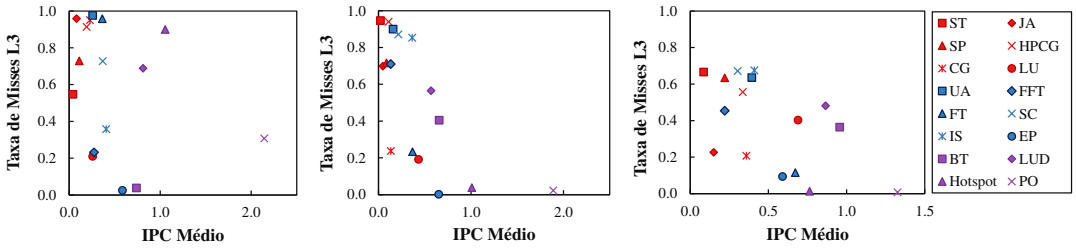
Oito kernels e pseudo-aplicações do NAS Parallel Benchmark [Bailey et al. 1991]: *BT*, *CG*, *EP*, *FT*, *IS*, *LU*, *SP* e *UA*. **Três aplicações da suíte de benchmarks do Rodinia** [Che et al. 2009]: *Hotspot*, *LUD* e *Streamcluster*. **Cinco aplicações de diferentes domínios**: *FFT*, *HPCCG*, *Método de Jacobi*, *Poisson*, e *Stream*.

As aplicações foram executadas com o conjunto de entrada pré-definido em cada suíte: As aplicações do NAS com o conjunto de entrada da classe "C"; As aplicações do Rodinia com o conjunto de entrada padrão disponibilizada junto com a suíte; E as demais aplicações com tamanho médio, de acordo com as características de cada aplicação. As aplicações escolhidas possuem diferentes comportamentos com relação ao acesso a memória e utilização do processador. Neste sentido, nós caracterizamos as aplicações de acordo com a taxa de *misses* na cache L3 e o IPC médio, conforme mostrado na Figura 2. Estas aplicações são representativas para o nosso experimento pois possuem diferentes características com relação ao acesso à memória principal e uso do processador (IPC médio): aplicações com baixo/médio/alto IPC e aplicações com baixo/médio/alto número de *misses* na cache L3. Os dados para esta classificação foram retirados diretamente dos contadores de *hardware* através do AMDuProf e do Intel *Performance Counter Monitor*.

3.2. Ambiente de Execução

Os experimentos foram realizados em três arquiteturas multicores, conforme mostrado na Tabela 1: AMD Ryzen 7 2700, AMD Ryzen 9 3900X e Intel Xeon E5 2640-v2[1].

[1]Alguns experimentos deste trabalho utilizaram os recursos da infraestrutura PCAD, http://gppd-hpc.inf.ufrgs.br, no INF/UFRGS



a) AMD Ryzen 7 2700 b) AMD Ryzen 9 3900 c) Intel Xeon E5 2640 v2

Figura 2. Características de cada aplicação em cada arquitetura *multicore*

Tabela 1. Principais características de cada arquitetura

	AMD Ryzen 7 2700	AMD **Ryzen 9 3900X**	**Intel Xeon E5 2640 v2**
Microarquitetura	Zen 2	Zen 2	Ivy Bridge-EP
Número de núcleos	8	12	16
Número de threads	16	24	32
Cache L3 (total)	16MB	64MB	20MB
Memória RAM	16GB	32GB	64GB
Frequências da CPU	1.55GHz, 2.8GHz, 3.2GHz	2.2GHz, 3.2GHz 3.8GHz	1.2GHz, 1.3GHz, 1.4GHz 1.5GHz, 1.6GHz, 1.7GHz 1.8GHz, 1.9GHz, 2.0GHz

Nós usamos o Sistema Operacional Linux Ubuntu com *kernel* v. 4.19.0. Cada aplicação foi compilada com GCC/G++ 9.2, usando a *flag* de otimização -O3. Cada aplicação foi executada com o número de *threads* igual ao número máximo de *threads* disponíveis no sistema e o Turbo Boost desativado, que é a maneira padrão como aplicações paralelas são executadas em arquiteturas *multicore*. As seguintes configurações de *governor* de DVFS foram avaliadas em cada arquitetura: *powersave*, *performance*, *ondemand*, *conservative* e *userspace*. Neste último, todos os níveis de frequência disponíveis no sistema foram utilizados (conforme ilustrado na Tabela 1). Adicionalmente, os *governors* foram executados com o Turbo Boost/CORE ativado e desativado.

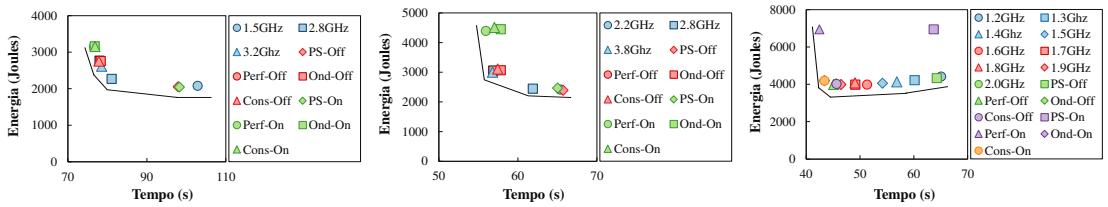
O EDP da execução de cada aplicação foi obtido pela multiplicação do tempo de execução (em segundos) pelo consumo de energia (em Joules). O tempo foi obtido através da função do OpenMP `omp_get_wtime`. Por outro lado, o consumo de energia foi obtido diretamente dos contadores de *hardware* presentes nos processadores modernos. No caso do processador Intel Xeon, a biblioteca *Running Average Power Limit* (RAPL) foi usada [Hähnel et al. 2012], enquanto que a biblioteca *Application Power Management* foi usada para o processador AMD Ryzen [Hackenberg et al. 2013]. Os resultados apresentados na Seção 4 são uma média de dez execuções com um desvio padrão menor que 0.5%.

4. Resultados

Nesta seção, inicialmente apresentamos e discutimos a exploração de espaço e projeto considerando diferentes frequências do processador e *governors* do DVFS (Seção 4.1). Então, na Seção 4.2, descrevemos PampaFreq.

4.1. Exploração de Espaço e Projeto

A Figura 3 apresenta os resultados de tempo de execução e consumo de energia para os três processadores quando cada aplicação é executada com cada nível de frequência de



(a) AMD Ryzen 7 2700 (b) AMD Ryzen 9 3900x (c) Intel Xeon E5 2640-v2

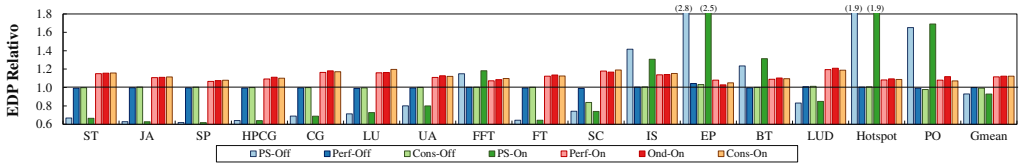
Figura 3. Tempo de execução e consumo de energia (média de todas as aplicações) em cada arquitetura.

operação do processador, *governors* do DVFS e técnica de *boosting* ativada/desativada. Os resultados são apresentados na forma de média geométrica de todas as aplicações executadas com cada configuração (e.g., frequência fixa em um único valor durante a execução da aplicação, *governor* DVFS e técnica de *boosting*). A legenda está organizada como segue: PS (*powersave*), Perf (*performance*), Ond (*ondemand*) e Cons (*conservative*); Off/On (técnica de *boosting* desativada/ativada). Para facilitar a visualização do gráfico e melhor identificar qual configuração apresenta melhor desempenho e consumo de energia, inserimos uma linha para representar a fronteira de Pareto.

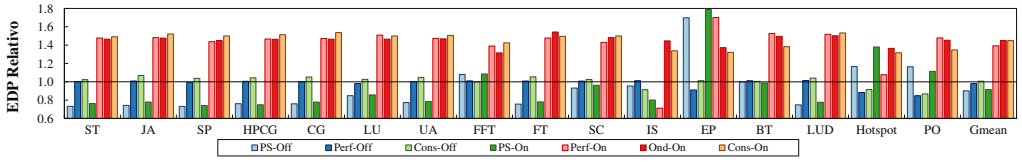
Conforme podemos observar nas Figuras 3a e 3b, para os processadores AMD, as configurações que entregam o melhor desempenho (menor tempo de execução) são também as que apresentam o maior consumo de energia (*Perf-On*). Isto é porque enquanto a frequência de operação maior devido ao Turbo CORE ativo propicia uma maior número de instruções executadas por segundo, ela também aumenta significativamente a potência, elevando o consumo de energia. Por outro lado, o menor consumo de energia implica num maior tempo de execução (*PS-Off* e *PS-On*) uma vez que a frequência de operação será menor. Para o processador Intel Xeon, conforme a frequência de operação aumenta, o tempo de execução também é reduzido. No entanto, como a diferença entre cada nível de frequência é significativamente menor que no AMD (no Intel cada nível aumenta em 100MHz). Por fim, o comportamento de consumo de energia e desempenho segue similar ao do processador AMD.

Na Figura 4, nós apresentamos os resultados de EDP das execuções de cada aplicação com cada *governor* do DVFS. O EDP é normalizado pelo resultado obtido com o *governor ondemand* com a técnica de *boosting* desativada, que é a maneira padrão utilizada na maioria dos Sistemas Operacionais *Linux*. Para o processador Intel (Figura 4c, o *governor ondemand* com a técnica de *boosting* desativada é capaz de entregar o melhor EDP para a maioria das aplicações e considerando a média geométrica. Por outro lado, para os processadores AMD (Figuras 4a e 4b), o *governor ondemand* falha em proporcionar o melhor custo-benefício entre desempenho e consumo de energia (e.g., EDP): para aplicações com baixo IPC e alta taxa de *misses* na cache L3, como por exemplo, ST, JA e SP, o governor *powersave* apresenta melhor EDP; enquanto que para aplicações com alto IPC (e.g., IS, LUD, Hotspot e PO), os *governors performance* e *conservative* com a técnica de *boosting* desativada apresentaram os melhores resultados de EDP.

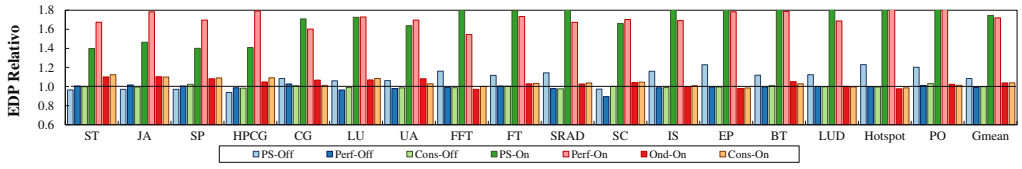
O comportamento observado nos processadores AMD está relacionado a quantidade de níveis de frequências disponíveis. Nos dois processadores alvo (e na grande maioria dos últimos processadores AMD), existem apenas três níveis de frequência: baixo, médio e alto. Portanto, o *governor ondemand* tem pouco espaço para atuar, reduzindo



(a) AMD Ryzen 7 2700



(b) AMD Ryzen 9 3900x



(c) Intel Xeon E5 2640-v2

Figura 4. EDP da execução de cada aplicação normalizado pelo resultado do *ondemand* com Turbo Boost/Core desativado.

as suas possibilidades de melhorar a utilização dos recursos computacionais. O resultado deste cenário é o aumento no EDP de acordo com as características das aplicações, conforme discutido anteriormente. Por outro lado, isto não ocorre nos processadores Intel devido a grande quantidade de níveis de frequência (vide Tabela 1). Neste sentido, na próxima seção nós apresentamos PampaFreq, para otimizar o EDP quando aplicações paralelas estão sendo executadas nos processadores AMD através da seleção ideal da frequência do processador e modo de operação da técnica de *boosting*.

4.2. Otimizando o EDP de Aplicações Paralelas nos Processadores AMD

Conforme discutido anteriormente, o *governor ondemand* não é capaz de fornecer a melhor utilização do sistema computacional nos processadores AMD. A principal justificativa para isto é a quantidade pequena de níveis de frequências disponíveis que o DVFS pode atuar. Neste sentido, para aplicações com baixo IPC e alto número de acessos à memória compartilhada, o *governor powersave* apresenta os melhores resultados, enquanto que para aplicações com alto IPC, o *governor performance* apresenta o melhor EDP. Portanto, nenhum *governor* disponível no sistema é capaz de adaptar a frequência de operação e modo de operação da técnica de *boosting* independente das características da aplicação e assim oferecer o melhor EDP. Neste sentido, nós apresentamos PampaFreq.

PampaFreq é uma metodologia composta de duas fases. A primeira fase utiliza algoritmos de aprendizado de máquina para encontrar padrões entre as métricas de *hardware* (e.g., IPC, acessos à memória cache L3 e principal), frequência de operação e o EDP resultante. Nesta etapa, nós implementamos um algoritmo de regressão linear múltipla com o objetivo de encontrar a relação entre as variáveis (IPC, acessos à memória, frequência e EDP) do modelo. Para calcular a regressão, nós selecionamos as aplicações com resultados mais significativos apresentados na Seção 4.1: ST, JA, FT, SC, Hotspot e

Algorithm 1 Algoritmo de Ajuste da Frequência de CPU e modo Turbo

Input: $F \leftarrow \{F_1, F_2, \ldots, F_k\}$: conjunto de frequência
$\theta \leftarrow$ Resultado da regressão linear múltipla de dados da primeira fase.
$\beta \leftarrow$ Intervalo entre medições.
$\delta \leftarrow$ Threshold para ativar/desativar modo Turbo.

```
1:  while isRunning(application) do
2:      φ ← get(IPC)
3:      Ω ← get(MEM_info)
4:      α ← findIdealFreq(φ, F, θ, Ω)
5:      setFreq(α)
6:      if (α = maxFreq and Ω ≥ δ) then
7:          setTurbo(on)
8:      else
9:          setTurbo(off)
10:     end if
11:     sleep(β)
12: end while
```

PO. Os dados resultantes desta etapa são então armazenados em um arquivo para posterior uso do algoritmo de otimização, conforme discutido abaixo.

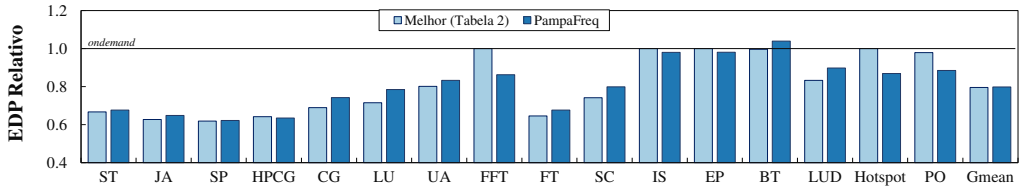
A segunda fase do PampaFreq consiste na execução do algoritmo de otimização definido em Algoritmo 1. O Algoritmo 1 recebe como entrada: o conjunto de frequências $F = \{F_1, F_2, ..., F_k\}$, os dados do arquivo resultante da análise realizada na primeira fase: resultado da regressão linear múltipla, o *threshold* para ativar/desativar a técnica de *boosting* dado o comportamento de memória da aplicação, além do intervalo entre medições das informações do *hardware*. Nesta última variável de entrada, quanto menor for o tempo de intervalo, maior será o *overhead* do algoritmo de modificação da frequência e modo de operação da técnica de *boosting*. Por outro lado, quanto maior for o tempo de intervalo, menor será a adaptabilidade do sistema. Portanto, através de testes experimentais, encontramos que o intervalo de 2 segundos entre as medições e tomadas de decisões do algoritmo apresentou os melhores resultados.

Portanto, enquanto a aplicação alvo está executando, o algoritmo proposto obtém o IPC médio e informações do sistema de memória (e.g., *hit/misses* na *cache* L2 e L3) do intervalo definido em β. Ambas as informações são obtidas diretamente dos contadores de *hardware* através do Linux *perf tools*, que está disponível na maioria das distribuições Linux. Uma vez que os dados são obtidos, (linhas 2 e 3), os mesmos são utilizados para (*i*) encontrar a frequência ideal entre as disponíveis no sistema, na linha 4. A função *findIdealFreq* aplica a função de *clustering* e seleção para encontrar a frequência ideal dada as características de IPC e comportamento de memória que reduzem o EDP da aplicação. Uma vez encontrada a frequência ideal, ela é setada nos núcleos através do comando *cpufreq* (linha 5). Por fim, se a frequência selecionada pela função *findIdealFreq* é a maior do sistema, então é verificado a necessidade de habilitar a técnica de *boosting* de acordo com o uso da memória (δ) entre as linhas 6 e 10. Por fim, durante o intervalo de tempo β, o sistema ficará coletando as informações do *hardware* para a próxima tomada de decisão.

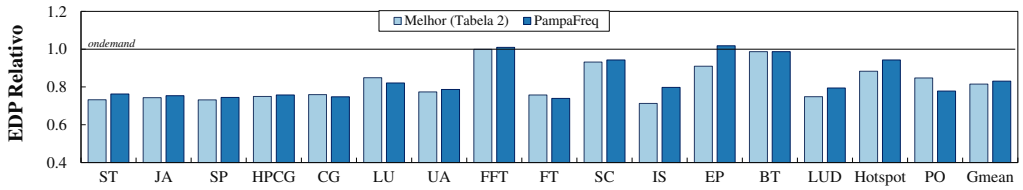
Os resultados da aplicação do PampaFreq sobre o conjunto de *benchmarks* descrito na Seção 3 são apresentados na Figura 5. Cada gráfico apresenta o EDP obtido durante a execução da aplicação com o algoritmo implementado por PampaFreq ativo (barra PampaFreq) e o melhor resultado obtido para cada aplicação durante a Seção 4.1 (barra Melhor) e descrito na Tabela 2. O EDP de cada configuração é normalizado pela execução com o *governor ondemand* e com a técnica de *boosting* desativada. Comparando os resul-

Tabela 2. Melhor configuração (Freq. e *governor* com ou sem Turbo CORE) para cada processador e aplicação.

	ST	*JA*	*SP*	*HPCG*	*CG*	*LU*	*UA*	*FFT*	*FT*	*SC*	*IS*	*EP*	*BT*	*LUD*	*Hotspot*	*PO*
AMD Ryzen 7	1.55	1.55	2.8	1.55	1.55	PS-On	PS-On	2.8	PS-On	1.55	Ond-Off	Ond-Off	2.8	PS-Off	2.8	2.8
AMD Ryzen 9	PS-Off	PS-Off	2.2	PS-On	2.2	2.8	2.2	2.8	PS-Off	PS-Off	Perf-On	3.8	2.8	2.8	3.8	Perf-Off



(a) AMD Ryzen 7 2700



(b) AMD Ryzen 9 3900x

Figura 5. EDP da execução de cada aplicação normalizado pelo resultado do *ondemand* com Turbo Boost/Core desativado.

tados obtidos pelo PampaFreq com o *governor ondemand*, o EDP é reduzido em até 38% no processador AMD Ryzen 7 e 26% no processador AMD Ryzen 9 para a aplicação SP. Considerando a média geométrica de todo o conjunto de aplicações, o EDP foi reduzido em 21% e 17% nos processadores AMD Ryzen 7 e 9, respectivamente.

Existem, ainda, alguns casos em que PampaFreq apresenta melhor EDP do que o melhor resultado encontrado pela exploração de espaço e projeto apresentado na Seção 4.1. Alguns exemplos compreendem as aplicações FFT, IS, EP, Hotspot e PO no AMD Ryzen 7; e FT e PO no AMD Ryzen 9. Este comportamento se dá pela capacidade que o algoritmo implementado pelo PampaFreq tem de (*i*) ativar e desativar o modo de operação da técnica de *boosting* em tempo de execução de acordo com as características da aplicação (e.g., acessos à memória principal) e (*ii*) considerar as características de uso do processador (IPC) e memória para definir a frequência ideal de operação do processador para cada instante da execução da aplicação. Nesta comparação com o melhor resultado encontrado pela execução da aplicação com a mesma frequência de operação ou *governor DVFS* do começo ao fim, no caso mais significativo (aplicação FFT no AMD Ryzen 7), o algoritmo implementado por PampaFreq reduziu o EDP em 14%.

5. Trabalhos Relacionados

[Lee 2009] propõem duas estratégias para diminuir o consumo de energia em execuções de aplicação em tempo real utilizando o DVFS. Utilizando de um algorítimo heurístico para classificar as aplicações de acordo com o grau de paralelismo. Então as aplicações classificadas como altamente paralelas são executadas em paralelo, enquanto as restantes são executas em apenas um núcleo. A segunda solução tem como princípio mover aplicações de núcleos que são pouco utilizados para os núcleos mais ativos e após isso desativar os núcleos que não estão sendo utilizados pela aplicação. [Raghavan et al. 2012]

propõem um *sprint* computacional para melhorar a responsividade das aplicações executadas em processadores de dispositivos móveis através do ajuste de frequência do processador e grau de exploração do paralelismo.

[Juan et al. 2013] propõem um algoritmo que visa melhorar o desempenho das aplicações com mínimo impacto no consumo de energia. A proposta aplica um algoritmo de aprendizado de máquina que analisa a relação entre o aumento da frequência e a dissipação de potência. Então, em tempo de execução o algoritmo usa uma heurística que determina os níveis de frequência e tensão. Autoturbo [Lo and Kozyrakis 2014] é um *daemon* que utiliza aprendizado de maquina para determinar o estado das técnicas de *boosting* durante a execução de uma aplicação. Primeiramente, o algoritmo executa as aplicações com diferentes números de *threads* e frequências para predizer suas características tais como comunicação e porcentagem de acessos a memória. A partir dessa predição o algoritmo se utiliza de uma heurística que decide o estado do *boosting* em tempo de execução.

[Maiti et al. 2015] propõem um framework para melhorar o consumo de energia e desempenho utilizando técnicas de seleção de núcleos, mapeamento de *threads* e o Turbo Boost. O algoritmo seleciona um grupo de núcleos do processador de acordo com a sua organização no chip e mapeia as *threads* com maior IPC para eles. Então, utilizando-se de modelos matemáticos, o *framework* determina, para cada núcleo, qual a frequência ideal para execução da aplicação. [Hsieh et al. 2015] propõem um *governor* para DVFS considerando o comportamento do uso de memória em jogos em dispositivos móveis. A proposta leva em consideração carga de trabalho executada na CPU e GPU e a porcentagem de acessos à memória para executar o gerenciamento das frequências. O *governor* executa uma verificação para decidir a frequência atual a cada vez que um novo quadro é mostrado na tela. O Adaptive Turbo Boosting [Kondguli and Huang 2018] é uma arquitetura que tem o objetivo de melhorar a performance por meio do uso de *boosting* e *look-aheed threading*. [Basmadjian and de Meer 2018] propõem um modelo de Markov para analisar *governors* que se baseiam na carga de trabalho do processador para gerenciamento de frequências como o *Ondemand* e o *Conservative*. A proposta considera diferentes frequências e o tempo de transição entre elas para calcular a eficiência energética e tempo de resposta desses *governors*.

Diferentemente dos trabalhos apresentados, nosso artigo realizou uma análise extensa considerando o espaço de exploração e projeto de aplicações paralelas com distintas características em diferentes arquiteturas *multicore*. Através desta análise, mostramos que os *governors* padrões do DVFS não são capazes de oferecer o melhor custo-benefício entre desempenho e consumo de energia nos processadores AMD. Dito isso, apresentamos PampaFreq para otimizar (*i*) a frequência de operação de cada núcleo e (*ii*) o uso da técnica de *boosting* em processadores AMD.

6. Conclusão

Este artigo apresentou uma análise da exploração de espaço e projeto da execução de dezesseis aplicações paralelas com diferentes níveis de frequência de operação do processador, *governors DVFS* e modos de operação de técnica de *boosting* em três plataformas *multicore* (Intel e AMD). Dada a limitação observada nos *governors* DVFS nos processadores AMD, este artigo apresentou PampaFreq, uma abordagem que seleciona o melhor

nível de frequência de operação da CPU e modo de operação do modo Turbo em processadores AMD. Resultados mostraram que o PampaFreq melhora o EDP em até 38% quando comparado ao *governor ondemand*, que é padrão nas distribuições Linux. Como trabalhos futuros, pretendemos investigar diferentes técnicas de economia de energia (e.g., *power-gating*, *clock-gating*) e sua aplicabilidade em arquiteturas heterogêneas.

Referências

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1991). The nas parallel benchmarks & summary and preliminary results. In *ACM/IEEE SC*, pages 158–165, USA. ACM.

Basmadjian, R. and de Meer, H. (2018). Modelling and analysing conservative governor of dvfs-enabled processors. In *Int. Conf. on Future Energy Systems*, pages 519–525.

Branover, A., Foley, D., and Steinman, M. (2012). Amd fusion apu: Llano. *Ieee Micro*, 32(2):28–37.

Charles, J., Jassi, P., Ananth, N. S., Sadat, A., and Fedorova, A. (2009). Evaluation of the intel® core™ i7 turbo boost feature. In *IISWC*, pages 188–197.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IEEE Int. Symp. on Workload Characterization*, pages 44–54, DC, USA. IEEE Computer Society.

dos Santos Marques, W., de Souza, P. S. S., Lorenzon, A. F., Beck, A. C. S., Rutzig, M. B., and Rossi, F. D. (2017). Improving edp in multi-core embedded systems through multidimensional frequency scaling. In *IEEE ISCAS*, pages 1–4. IEEE.

Hackenberg, D., Ilsche, T., Schone, R., Molka, D., Schmidt, M., and Nagel, W. E. (2013). Power measurement techniques on standard compute nodes: A quantitative comparison. In *ISPASS*, pages 194–204.

Hähnel, M., Döbel, B., Völp, M., and Härtig, H. (2012). Measuring energy consumption for short code paths using rapl. *SIGMETRICS Perf. Evaluation Rev.*, 40(3):13–17.

Haj-Yahya, J., Mendelson, A., Ben Asher, Y., and Chattopadhyay, A. (2018). *Power Management of Modern Processors*, pages 1–55. Springer Singapore, Singapore.

Hsieh, C.-Y., Park, J.-G., Dutt, N., and Lim, S.-S. (2015). Memory-aware cooperative cpu-gpu dvfs governor for mobile games. In *IEEE ESTIMedia*, pages 1–8. IEEE.

Juan, D.-C., Garg, S., Park, J., and Marculescu, D. (2013). Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In *Int. Conf. on Hardware/Software Codesign and System Synthesis*, page 8. IEEE Press.

Kondguli, S. and Huang, M. (2018). A case for a more effective, power-efficient turbo boosting. *ACM Trans. on Architecture and Code Optimization*, 15(1):1–22.

Lee, W. Y. (2009). Energy-saving dvfs scheduling of multiple periodic real-time tasks on multi-core processors. In *Int. Symp. on Distributed Simulation and Real Time Applications*, pages 216–223.

Lo, D. and Kozyrakis, C. (2014). Dynamic management of turbomode in modern multi-core chips. In *Int. Symp. on High Performance Computer Architecture*, pages 603–613. IEEE.

Lorenzon, A. F. and Beck, A. C. S. (2019). *Parallel Computing Hits the Power Wall - Principles, Challenges, and a Survey of Solutions*. Springer Briefs in Computer Science. Springer.

Lorenzon, A. F., Dellagostin Souza, J., and Schneider Beck, A. C. (2017). Laant: A library to automatically optimize edp for openmp applications. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1229–1232.

Lorenzon, A. F., Sartor, A. L., Cera, M. C., and Schneider Beck, A. C. (2015). Optimized use of parallel programming interfaces in multithreaded embedded architectures. In *2015 IEEE Computer Society Annual Symposium on VLSI*, pages 410–415.

Maiti, S., Kapadia, N., and Pasricha, S. (2015). Process variation aware dynamic power management in multicore systems with extended range voltage/frequency scaling. In *IEEE Int. Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4. IEEE.

Marques, S. M. V. N., Medeiros, T. S., Rossi, F. D., Luizelli, M. C., Girardi, A. G., Beck, A. C. S., and Lorenzon, A. F. (2019). The impact of turbo frequency on the energy, performance, and aging of parallel applications. In *27th IFIP/IEEE International Conference on Very Large Scale Integration, VLSI-SoC 2019, Cuzco, Peru, October 6-9, 2019*, pages 149–154. IEEE.

Raghavan, A., Luo, Y., Chandawalla, A., Papaefthymiou, M., Pipe, K. P., Wenisch, T. F., and Martin, M. M. (2012). Computational sprinting. In *IEEE international symposium on high-performance comp architecture*, pages 1–12. IEEE.

Sartor, A. L., Lorenzon, A. F., Carro, L., Kastensmidt, F., Wong, S., and Beck, A. C. S. (2017). Exploiting idle hardware to provide low overhead fault tolerance for vliw processors. *J. Emerg. Technol. Comput. Syst.*, 13(2).

Wamhoff, J.-T., Diestelhorst, S., and Fetzer, C. The turbo diaries: Application-controlled frequency scaling explained.