Avaliação de um framework de apoio ao desenvolvimento de heurísticas de escalonamento sensível ao consumo energético

Bruno Giacobo Pinto[1]*, **Lucas M. S. Xavier**[1]†‡, **Gerson Geraldo Homrich Cavalheiro**[1]

[1]Centro de Desenvolvimento Tecnológico - CDTec/Computação - UFPel
Campus Porto - Rua Gomes Carneiro 1 - Pelotas (RS) - Brasil

{bgpinto, lmdsxavier, gerson.cavalheiro}@inf.ufpel.edu.br

Abstract. *Modern processors provide some information about their energy consumption. However, the lack of standardization and uniformization of such data narrows the scope of solutions that are built upon that kind of information. This paper presents a framework that offers a common interface of service in order to monitor processor energetic consumption at run-time in a uniform fashion. As a validation step, the energy consumption of concurrent applications was monitored. The applicability of the developed tool in a energy-aware scheduling scheme is discussed.*

Resumo. *Os processadores modernos oferecem informações sobre seu consumo de energia. No entanto, a inexistência de uniformização e padronização desses dados restringe a portabilidade de soluções que os utilizam. Este trabalho apresenta um framework que provê uma interface única de serviços para acessar as informações de consumo energético de processadores de forma uniforme em tempo de execução. Como validação, o consumo energético de aplicações concorrentes é monitorado. Um estudo de casos discute a aplicabilidade dessa ferramenta em uma estratégia de escalonamento sensível ao consumo energético.*

1. Introdução

Como alternativa às limitações tecnológicas para produção de processadores com altas frequências de operação, as arquiteturas *multicore* foram desenvolvidas e tornaram-se elemento presente em praticamente todas as soluções computacionais para o processamento de alto desempenho. Essa tecnologia é atraente, uma vez que suporta a execução eficiente de um vasto leque de aplicações a um custo abordável. No entanto, caso nenhuma estratégia de escalonamento do uso das unidades de execução seja adotada, a energia consumida por essas plataformas pode fazer com que seu custo operacional seja maior que o custo de sua aquisição [Yang et al. 2013].

Diversas abordagens foram propostas para otimizar a execução de aplicações concorrentes e reduzir o consumo de energia. Em nível de hardware, técnicas que manipulam a frequência e a voltagem dos processadores, como *Dynamic Voltage and Frequency Scaling (DVFS)*, ou desligam partes ociosas desse hardware, como *Clock Gating*,

*Bolsista IC/PROCAD/CAPES

†Bolsista PIBIC/CNPq

‡O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

são empregadas [Venkatachalam and Franz 2005]. Em nível de software, a utilização de heurísticas presentes no *kernel* dos sistemas operacionais é uma abordagem recorrente [Pallipadi and Starikovskiy 2006, Chu et al. 2013]. Recentemente, estratégias de escalonamento sensíveis ao consumo energético (*energy-aware* em inglês) vêm sendo utilizadas em nível aplicativo [Libutti et al. 2014, Petrucci et al. 2015], onde tarefas no contexto do programa em execução são mapeados sobre os recursos da arquitetura visando otimizar tempo ou consumo de energia.

Seja para uma etapa de validação, seja para tomada de decisões, aplicações *energy-aware* necessitam de dados de consumo de energia. As modernas arquiteturas de processadores, como Sandy Bridge (Intel) e Bulldozer (AMD), oferecem interfaces que contabilizam eventos de hardware e disponibilizam uma estimativa de consumo energético do processador. As vantagens de utilizar tais interfaces estão associadas ao fato do programa poder obter, em tempo de execução, dados atuais sobre o consumo, podendo administrar de forma dinâmica a taxa de execução em função do consumo observado. As dificuldades estão associadas à inexistência de um padrão para apresentação destas informações pelas diferentes interfaces, uma vez que não há padronização nem para o conjunto de serviços oferecido, nem para a representação das grandezas de consumo aferidas.

Diante da pluralidade de mecanismos de monitoramento, da escassez de ferramentas de mais alto nível para obtenção de dados de consumo e da demanda por soluções de escalonamento *energy-aware*, este trabalho apresenta um *framework* que oferece um conjunto de serviços para acessar dados de consumo fornecidos pelas interfaces RAPL (Intel) [Intel 2013] e APM (AMD) [AMD 2013] e ainda permite gerenciar a frequência de operação individual a cada núcleo de processamento em uso. O objetivo é permitir o desenvolvimento de software *energy-aware* portável entre as diferentes plataformas, uma vez que o *framework* oferece uma interface única para acesso aos dados de consumo e também se responsabiliza pela conversão das grandezas representando o consumo energético. Uma validação do uso desse *framework* é apresentado, uma vez que RAPL e APM fornecem, como dados de consumo, energia em Joules (J) e potência em Watts (W), respectivamente. Como estudo de caso, é apresentada a discussão de um estratégia de escalonamento *energy-aware* em uma aplicação regular.

O restante deste artigo está organizado como segue. Na seção 2, alguns trabalhos relacionados são discutidos. A seção 3 ilustra uma representação formal para o consumo de energia de processadores. A seção 4 sumariza as principais metodologias de monitoramento de energia. A seção 5 apresenta o *framework* desenvolvido e, na seção 6, é feito uma análise do uso desse *framework* em aplicações regulares. A seção 7 discute a aplicação do *framework* em uma estratégia de escalonamento *energy-aware*. Por último, são abordadas algumas considerações finais na seção 8.

2. Trabalhos Relacionados

De uma forma geral, entende-se que melhorar o desempenho de um programa consiste em reduzir seu tempo total de execução, quando se considera tempo de processamento como unidade de medida. No entanto, todos os núcleos de processamento de uma arquitetura multiprocessada não necessitam estar operando em sua frequência máxima durante toda a execução de um programa paralelo [Patki et al. 2013, Lu et al. 2000] para que o menor tempo de execução seja obtido. Na hipótese motivadora do presente trabalho, considera-se

que em uma aplicação regular [Gautier et al. 1995], o caminho crítico de uma aplicação [Graham 1972] dita o limite do desempenho obtido, sendo que toda execução fora deste caminho crítico é elegível para execução sobre uma unidade de processamento com frequência (portanto, também consumo), reduzido.

Em [Baskiyar and Abdel-Kader 2010] é considerado o caso do escalonamento de programas descritos na forma de um grafo dirigido acíclico (DAG) em uma arquitetura heterogênea, com o objetivo de reduzir tanto o tempo total de processamento como o gasto energético. Neste trabalho, a heterogeneidade é marcada pela possibilidade dos processadores aumentarem e diminuírem, de uma forma discreta, a frequência de sua operação. O algoritmo é processado de forma estática da seguinte forma. Em um primeiro momento, uma estratégia de alocação, baseada em EFT (*Earliest Finish Time*), é realizada para distribuir as tarefas entre os processadores. Em uma segunda etapa, os processadores são visitados e a soma de seus custos é ponderada com a mínima frequência possível que não aumente o *makespan* (tempo total de execução). Nas conclusões deste trabalho, os autores indicam que uma alta taxa de paralelismo da aplicação resulta em uma baixa redução no consumo energético.

Em [Shekar and Izadi 2010] há outra proposta de escalonamento de DAG, que estende a estratégia DLS (*Dynamic Level Scheduling*) para redução de tempo de execução e consumo energético. O algoritmo básico, aplicado de forma estática, considera a valorização do escalonamento de uma tarefa sobre o processador que resultar em menor consumo energético. Uma etapa de refino considera um fator de desempenho adicional, visando reduzir a penalização em termos de tempo de execução.

Considerando escalonamentos realizados em tempo real, durante a execução, encontramos o trabalho de [Houben and Halang 2014]. Baseado em uma estratégia EDF (*Earlierst Deadline First*), uma lista de tarefas é mantida ordenada pelo tempo requerido de término de cada tarefa. No algoritmo modificado, ordenação nesta lista considera as diferentes velocidades de processamento dos processadores para alocação das tarefas, considerando que, quanto menor a velocidade, menor o consumo energético gerado.

Uma estratégia de mapeamento entre *threads* da aplicação e *cores* voltado para redução do consumo de energia considerando novas arquiteturas *multicores* heterogêneas é apresentado em [Petrucci et al. 2015]. Um modelo de otimização baseado em programação linear é empregado para encontrar um mapeamento eficiente entre uma *thread* e um *core* de forma que tempo e consumo de energia sejam otimizados. Para compor esse modelo, contadores de performance que influenciam indiretamente no consumo energético de uma aplicação são utilizados. Nesse trabalho, mostrou-se que o uso de dados de consumo em tempo de execução permite que estratégias *energy-aware* alcancem reduções significativas na energia gasta por uma aplicação.

Neste trabalho, implementou-se, como estudo de caso, uma estratégia de escalonamento sensível ao consumo e voltada para aplicações regulares utilizando as funcionalidades do *framework* desenvolvido. Como premissa, buscou-se reduzir o consumo energético ao explorar o compromisso entre a variação de frequência e otimizações no mapeamento de tarefas que compõem o caminho critico da aplicação. Na estratégia proposta, alguns *cores* executando tarefas no caminho crítico tiveram sua frequência elevada ao máximo, enquanto os demais tiveram sua frequência reduzida.

3. Modelagem do Consumo Energético

Para entender como estratégias sensíveis ao consumo energético podem ser efetivas e como métodos de monitoramento de informações de consumo podem ser implementados, é necessário entender o relacionamento entre potência, energia e tempo. A energia consumida por um processador pode ser formalmente modelada segundo a equação [Sheikh et al. 2012]:

$$E = \int_{T} P \cdot \mathrm{d}t \tag{1}$$

Onde P representa a potência, E denota a energia consumida e T denota um intervalo de tempo contínuo. Pela equação 1, pode-se inferir que a energia consumida é o acúmulo do trabalho realizado pelo circuito ao longo do tempo. A potência define o valor médio de trabalho realizado ao longo desse intervalo.

$$P_{Dinâmica} = C_c \times V^2 \times f \tag{2}$$

Em processadores implementados com tecnologia CMOS (*Complementary Metal Oxide Semiconductor*), o consumo de potência tem duas origens: $P_{Estática}$ e $P_{Dinâmica}$. A primeira é consequência da tecnologia de implementação do circuito. A segunda, representada na equação 2, resulta da atividade de chaveamento do circuito, expressando, portanto, o trabalho efetivo realizado pelo mesmo. Técnicas de gerenciamento de recursos que visam reduzir o consumo energético, como DVFS, exploram essa relação para implementar suas estratégias. Nessa equação, C_c representa a capacitância do circuito e, portanto, é considerado uma constante. A variáveis f e V representam a frequência e a tensão de alimentação, respectivamente.

4. Métodos de Monitoramento de Energia

Os processadores modernos passaram a contar com contadores de performance, um mecanismo para contabilizar eventos e informações em tempo de execução. Explorando esses contadores, ferramentas baseadas em modelos estatísticos foram construídas para monitorar o consumo energético [Joseph and Martonosi 2001, Goel 2011]. Dados como número de *misses* na cache, número de operações de ponto flutuante e número de desvios condicionais, por exemplo, são acessados em nível de software e combinados para prover uma estimativa da energia consumida. Embora essa abordagem proporcione informações de granularidade mais fina e dispense um hardware para medição direta, ela requer uma etapa de ajuste de parâmetros para a máquina a ser monitorada. Ainda, essa metodologia pode apresentar imprecisões no dado aferido, caso o modelo não considere determinadas características da aplicação em execução [McCullough et al. 2011].

Em resposta à demanda por soluções eficientes de gerenciamento de energia, os fabricantes de processadores passaram a integrar, em suas arquiteturas, ferramentas que implementam modelos de estimativa para monitorar o consumo energético. A solução apresentada pela Intel, chamada RAPL (*Running Average Power Limit*), relaciona contadores de performance por meio de uma regressão linear para estimar a energia consumida pelo processador em intervalos de 1 milissegundo. Os dados estimados de consumo

são armazenados em registradores não arquiteturais que podem ser acessados pelo sistema operacional por instruções privilegiadas de leitura. A interface proposta pela AMD, chamada APM (*Application Power Management*), realiza uma amostragem dinâmica dos contadores de performance dos *cores* e os relaciona de modo a estimar a potência consumida para todo o processador dentro de um intervalo de tempo variável. Assim como a RAPL, essa ferramenta também armazena os dados de consumo em registradores para serem acessados pelo sistema operacional. Essas interfaces foram validadas em [Hähnel et al. 2012, Rotem et al. 2012, Jotwani et al. 2010].

5. O Framework

Para que soluções *energy-aware* portáveis e robustas sejam implementadas, faz-se necessário um mecanismo uniforme de aquisição de dados de consumo. Visando dar suporte à concepção de tais aplicações, propõe-se um *framework* capaz de expor informações de consumo energético por meio de uma interface única de serviços em tempo de execução. As informações de consumo (potência ou energia) disponibilizadas são provenientes das interfaces RAPL e APM e são acessadas com suporte do sistema operacional. A ferramenta desenvolvida prevê, ainda, um mecanismo que permite ao usuário variar a frequência de operação de *cores* do processador individualmente. Para garantir portabilidade, é necessário que o *framework* tenha conhecimento da topologia da arquitetura. Isso é alcançado utilizando-se a API (*Application Programming Interface*) Hwloc [Broquedis et al. 2010] que, em tempo de execução, expõe informações dos elementos de processamento presentes. Dessa forma, a ferramenta desenvolvida é capaz de selecionar entre RAPL ou APM e monitorar o consumo individual dos nós ou sockets do processador.

Devido às interfaces APM e RAPL exporem diferentes dados de consumo energético, uma etapa de conversão é necessária para que a medição seja independente de arquitetura. A ferramenta desenvolvida se encarrega dessa uniformização explorando a relação expressa na equação 1. Desde o início de sua execução, para cada nó, o *framework* efetua amostragens periódicas de tempo para possibilitar que a conversão entre potência média e energia seja realizada. Essas amostras são utilizadas para calcular a forma convertida do dado de consumo requisitado pelo usuário, pois a energia pode ser obtida integrando-se os valores de potências em um intervalo e a potência pode ser obtida derivando-se a energia em relação ao mesmo intervalo.

Utilizando as políticas de manipulação de frequência presentes no *kernel* Linux, implementou-se um mecanismo para alterar a frequência individual dos *cores* do processador. Por questões de portabilidade, a implementação corrente suporta duas variações: a modalidade *performance* eleva a frequência de operação de um *core* ao máximo, ao passo que a modalidade *powersave* atribui o menor valor possível de frequência ao *core* alvo. Visando facilitar a integração em aplicações *energy-aware*, implementou-se o *framework* em C++, seguindo a tendência de projeto de ferramentas de programação concorrente [Cavalheiro and Du Bois 2014].

6. Análise de Aplicação do Framework

Como etapa de avaliação, foi monitorado o consumo energético de três aplicações regulares intensivas em CPU visando ilustrar as funcionalidades do *framework*. A primeira aplicação é uma implementação *multithread* do algoritmo de fatoração LU aplicado à matrizes

quadradas. A segunda aplicação é uma versão paralela do algoritmo de aglomeração de grupos (AGC – *Agglomerative Clustering* em inglês). O último experimento consiste em um *benchmark* sintético que simula duas estratégias simples de mapeamento de tarefas sobre a arquitetura subjacente quando da execução de uma carga uniforme de trabalho.

Tabela 1. Configurações das Plataformas de Teste

Nome	Processador	Memória	Linux Ubuntu
Intel	Intel Core i7 de 3,6 GHz	8 GB	14.04 LTS
AMD	AMD FX-8320 de 3,4 GHz	16 GB	14.04 LTS

Como ferramenta de expressão de concorrência, tanto para o algoritmo de fatoração *LU*, quanto para o AGC, o ambiente de programação Anahy-3 [Araujo 2013] foi utilizado. Essa ferramenta, voltada para arquiteturas *multicore*, apresenta uma interface de programação similar a Pthreads [Nichols, Bradford and Buttlar, Dick and Farrell, Jacqueline 1996] e implementa uma estratégia de escalonamento baseada em roubo de trabalho. A Tabela 1 sumariza a configuração das plataformas utilizadas para a execução dos experimentos. A máquina *Intel* possui 4 núcleos físicos de processamento e suporte a *HyperThreading*, ao passo que a máquina *AMD* possui 8 núcleos físicos, ambas com 3 níveis de *cache*. As Figuras 1 e 2 sumarizam os resultados de tempo e energia para 30 execuções das aplicações LU e AGC, respectivamente. Para cada caso, variou-se o número de *threads* em um intervalo de 1 a 32, acrescendo-se esse numero em potências de 2.

6.1. Fatoração LU

A decomposição LU é um método numérico para fatoração de matrizes de forma que uma dada matriz A pode ser reescrita pela multiplicação das matrizes diagonal superior U (de *upper* em inglês) e diagonal inferior L (de *lower* em inglês) de A . Para obter as matrizes diagonais L e U, implementou-se uma versão concorrente do método de Crout [Press et al. 1988].

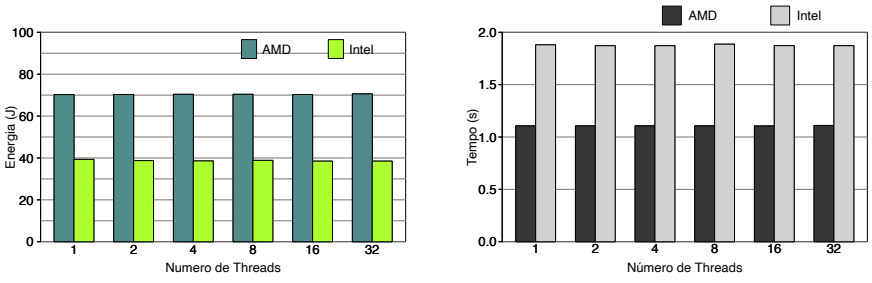

Figura 1. Resultados de tempo e energia para a *LU* nas máquinas *Intel* e *AMD*.

6.2. Agglomerative Clustering

Um algoritmo de aglomeração de grupos adota uma abordagem hierárquica para classificação e representação de grandes conjuntos de dados [Walter et al. 2008]. Esse algoritmo recebe como entrada um critério de similaridade e um conjunto de dados e retorna uma árvore binária cujos nós são grupos compostos por elementos que pertencem

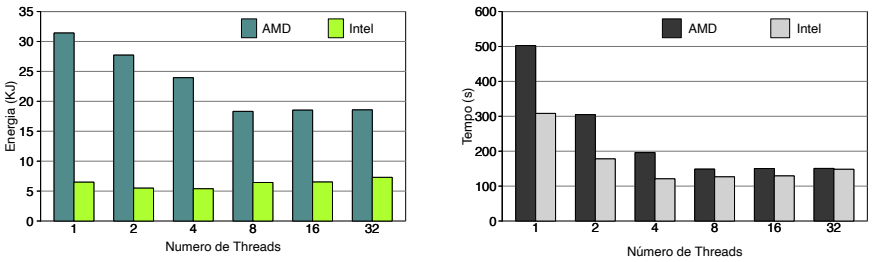

Figura 2. Resultados de tempo e energia para o *AGC* nas máquinas *Intel* e *AMD*.

a um mesmo nível de similaridade. Devido ao grande paralelismo apresentado pelo AGC, uma versão paralela foi implementada e seu consumo energético foi monitorado. Como conjunto de entrada, escolheu-se 50000 pontos aleatórios do plano cartesiano e definiu-se a similaridade entre esses pontos como a distância entre cada par de pontos.

6.3. Aplicação Sintética

Para simular diferentes padrões de consumo de energia do processador, implementou-se, em Pthreads, um *benchmark* que distribui uma carga sintética entre diferentes *cores* para induzir diferentes níveis de utilização das unidades de processamento. Esse comportamento é alcançado bloqueando-se *threads* por um período de tempo determinado dinamicamente segundo estatísticas de uso do sistema. A carga sintética calculada pelas unidades concorrentes da aplicação é dividida uniformemente no início da execução do *benchmark* e consiste de instruções aritméticas simples.

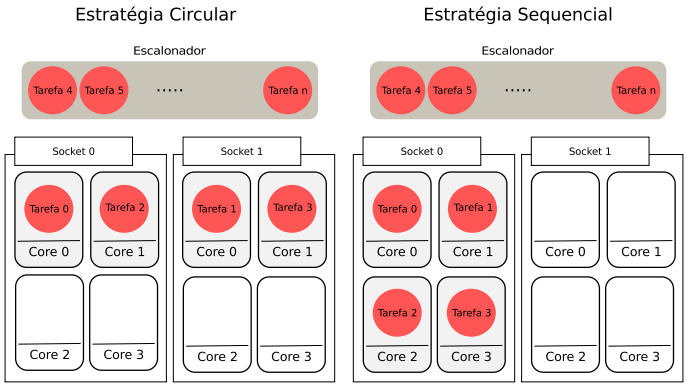

Figura 3. Distribuição de 4 tarefas seguindo as politicas *circular* e *sequencial*

Como plataforma de testes, a máquina que suporta a execução da aplicação sintética conta com 4 nós de processamento AMD Opteron 6276 de 2,3 GHz com 3 níveis de cache (L3 de 6MB, L2 de 2MB e L1 de 64KB). Cada nó ou socket dessa arquitetura contém 16 *cores* que acessam, de maneira não uniforme, 32GB de memória separados em blocos de 16GB, totalizando 128GB de RAM. Como suporte de software, a plataforma conta com o sistema operacional GNU/Linux Ubuntu 12.04 LTS de 64 bits com *kernel* versão 3.2.0-65. Para esse experimento, optou-se por utilizar uma arquitetura diferente, uma vez que ela favorece a implementação do mapeamento das cargas e auxilia na formulação de hipóteses sobre possíveis otimizações em estratégias de escalonamento intra-nós. O mesmo

experimento não pode ser reproduzido em uma arquitetura Intel devido a indisponiblidade de uma máquina com arquitetura equivalente.

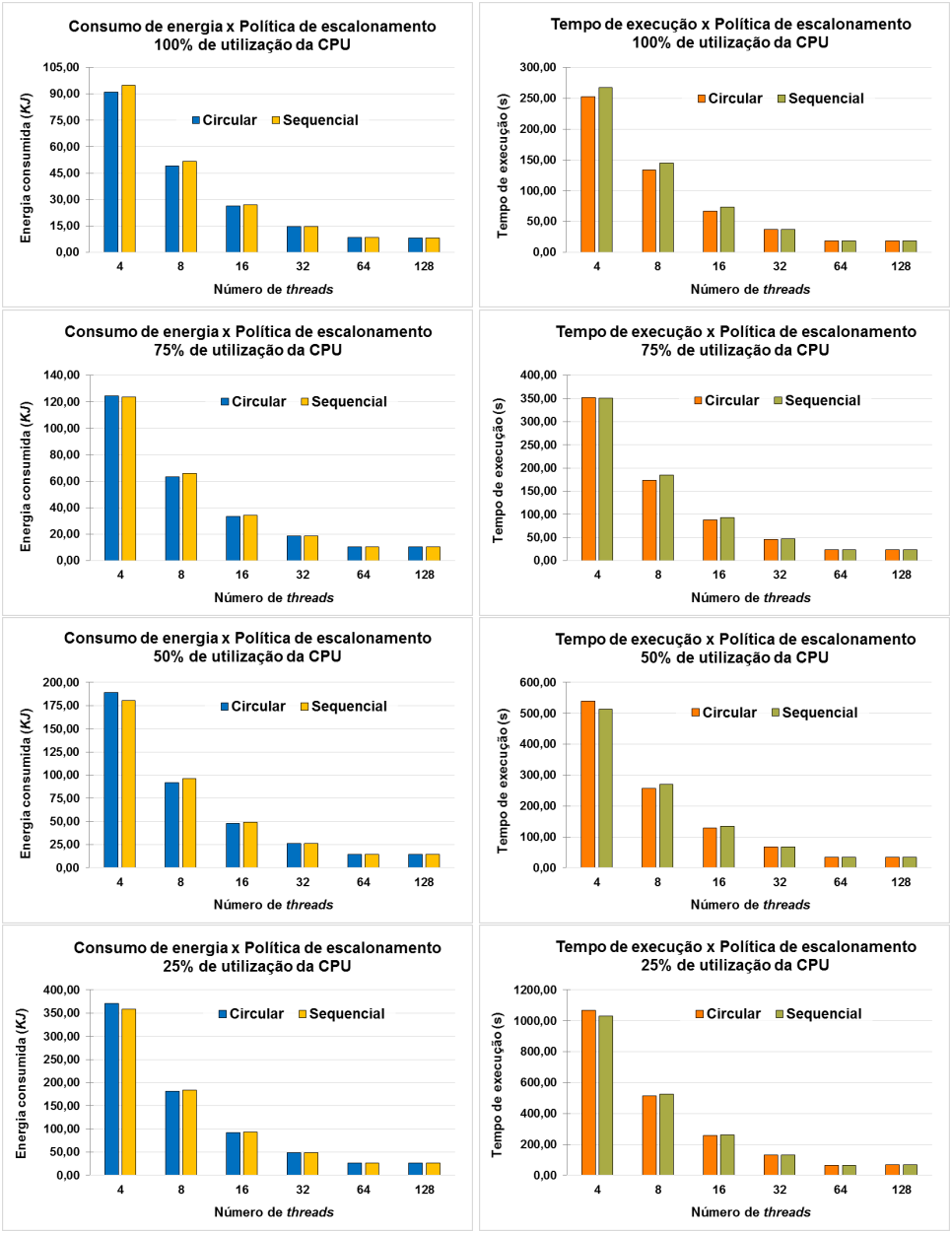

Figura 4. Resultados de tempo e energia para a aplicação sintética.

Visando observar variações no consumo de energia e no tempo de execução, a implementação da distribuição de carga da aplicação sintética obedece a duas políticas simples de mapeamento. Como mostra a Figura 3, a primeira estratégia, denominada *circular*, atribui tarefas de forma alternada entre aos nós de processamento disponíveis. Na segunda política, denominada *sequencial*, cada *core* que compõe um nó recebe uma tarefa e, quando todos os *cores* desse nó forem preenchidos, o próximo nó é selecionado.

Na Figura 4, os resultados de tempo e energia para 30 execuções do *benchmark* sintético são observados. Nesse experimento, variou-se o número de *threads* de 4 a 128 em

potências de 2, bem como a taxa de uso da CPU de 25% a 100% para as duas políticas de mapeamento. Pode-se verificar que, de modo geral, os valores de energia acompanharam os valores de tempo de execução. Em relação ao consumo de energia, um aumento pode ser observado quando recursos de processamento foram subutilizados. Ainda, a uma significância de 99%, nota-se que a distribuição *circular* resultou em uma execução 8,20% mais rápida e consumiu 5,13% menos energia para casos de 8 ou 16 *threads*. Isso é esperado, pois essa estratégia de mapeamento reduz o número de conflitos pelos recursos, implicando em um maior aproveitamento do hardware subjacente.

7. Estudo de Caso

É possível determinar, a priori, em uma aplicação regular as tarefas que compõem o caminho de maior custo (caminho crítico). Explorando essa característica, propõe-se uma estratégia de escalonamento *energy-aware* utilizando as funcionalidades do *framework*. Para implementação desse experimento, escolheu-se o ambiente de programação concorrente Anahy-3. Essa ferramenta lança tarefas para serem executadas invocando-se a diretiva *Fork* e sincroniza a execução dessas tarefas por meio de uma diretiva *Join*. No entanto, ao contrário de OpenMP [OpenMP ARB 2013], esse ambiente não impõe que tarefas sejam criadas de forma aninhada, ou seja, quando uma tarefa é gerada, esta pode ser sincronizada por qualquer tarefa e não só pela que a criou.

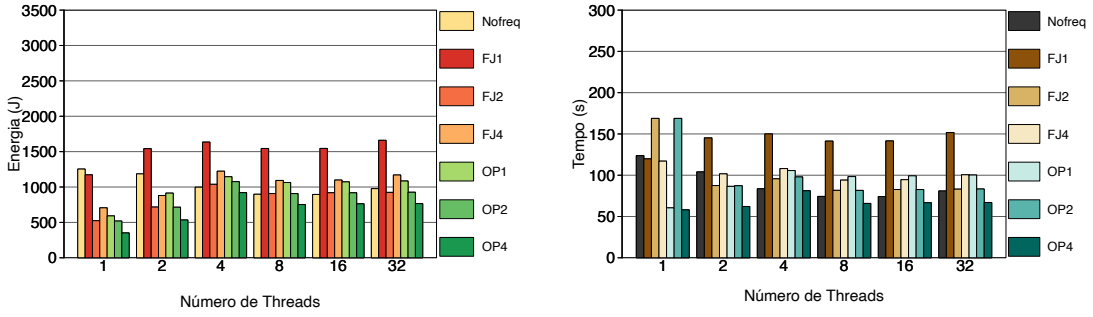

Figura 5. Resultados de tempo e energia para o Fibonacci na máquina *AMD*.

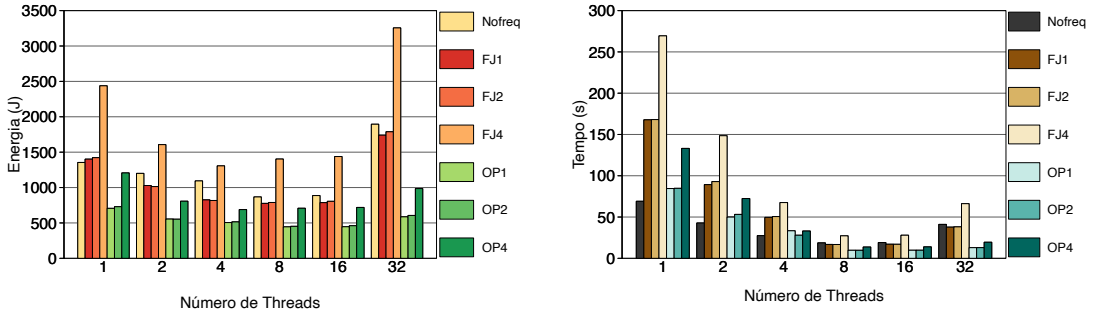

Figura 6. Resultados de tempo e energia para o Fibonacci na máquina *Intel*.

A estratégia de escalonamento proposta, chamada *Otimizado* (*OP*), funciona como segue: *cores* que compõem o caminho crítico da aplicação têm sua frequência elevada

ao máximo, os demais têm sua frequência reduzida ao valor mínimo. Ainda, alterou-se a forma como Anahy-3 gera tarefas, de forma que tarefas só são geradas em regime de aninhamento. A título de experimento, variou-se essa estratégia de forma que 1, 2 e 4 *cores* executando o caminho crítico tiveram sua frequência afetada. Denominou-se tais variações de *OP1*, *OP2*, e *OP4*, respectivamente. Como comparação, outros dois experimentos são propostos. No primeiro, denominado *Nofreq*, nem a estratégia de criação de tarefas, nem a frequência dos *cores* sofreram alterações. No segundo, denominado *Fork/Join*, alterou-se somente a frequência dos *cores*. Assim como em *OP*, três variações de frequência, denominadas *FJ1*, *FJ2* e *FJ4*, são avaliadas. Pelo fato de possuir uma estrutura regular, a aplicação escolhida para avaliar a estratégia *OP* consiste no cálculo recursivo do enésimo elemento da sequência de Fibonacci. As Figuras 5 e 6 apresentam os resultados médios de tempo de execução e consumo de energia para 30 execuções desse algoritmo.

Na máquina *AMD*, pode-se notar que, de modo geral, os comportamentos de tempo e energia foram similares. Comparando-se as estratégias *Nofreq* e *OP*, observou-se, a um nível de importância de 99%, uma economia média de 15% para os casos em que *threads* foram variadas de 4 a 32 por parte da estratégia *OP4*. Com 1 e 2 *threads*, observou-se que *OP4* foi capaz de reduzir, nesses casos, 71,95% e 54,89%, respectivamente, em relação a *Nofreq*. No entanto, as variações *OP1* e *OP2* consumiram mais energia que *Nofreq*, chegando a 19,86% no caso em que 16 *threads* foram utilizadas. Em relação a estratégia *Fork/Join*, a estratégia otimizada conseguiu reduzir em média 35,08% quando da comparação entre *FJ4* e *OP4* para todos as configurações de *threads*. O mesmo comportamento foi observado quando comparados *FJ1* e *OP1*, porém, para *FJ2* e *OP2* não houve diferença significativa. Na máquina *Intel*, a estratégia *OP* reduziu em média 52,83% mais energia em comparação a *Nofreq* em todos os casos, apesar do comportamento dos tempos de execução não terem convergido com o comportamento do consumo em alguns casos. Quando comparado a estratégia *Fork/Join*, a estratégia *OP* também obteve melhores resultados e manteve uma economia média de 47,14% para todas configurações. Não se observou, em nenhum dos experimentos, desvio padrão maior que 2%.

8. Conclusão

Apresentou-se, neste trabalho, um *framework* capaz de, em tempo de execução, monitorar de forma portável e uniforme a potência e energia das mais recentes famílias de processadores. As funcionalidades dessa ferramenta foram ilustradas no contexto do escalonamento de aplicações regulares. Discutiu-se, em um estudo de casos, a aplicabilidade da ferramenta desenvolvida no contexto de uma estratégia de escalonamento *energy-aware*. Avaliando-se esse experimento, pode-se notar que, de maneira geral, a estratégia proposta foi capaz de reduzir significativamente a energia consumida em relação as demais estratégias quando metade dos *cores* da arquitetura foram utilizados (*OP4* na máquina *AMD* e *OP2* na máquina *Intel*). Isso ocorre devido uma redução no *overhead* de criação de *threads* quando *Fork/Join* aninhado foi utilizado, prevenindo uma sobrecarga na execução dos *cores* que tiveram sua frequência de operação reduzida. Soma-se a isso o fato de que os *cores* que executam tarefas do caminho crítico tiveram suas frequências maximizadas, ocasionando uma redução no tempo de execução de modo geral.

Referências

AMD (2013). *AMD Family 15h Processor BIOS and Kernel Developer Guide*. Rev 3.14.

Araujo, A. S. (2013). Anahy-3: Um novo ambiente de execução otimizado para arquiteturas multicore. Trabalho acadêmico (graduação), Universidade Federal de Pelotas.

Baskiyar, S. and Abdel-Kader, R. (2010). Energy aware dag scheduling on heterogeneous systems. *Cluster Computing*, 13(4):373–383.

Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. (2010). hwloc: A generic framework for managing hardware affinities in HPC applications. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 180–186, Pisa. IEEE.

Cavalheiro, G. G. H. and Du Bois, A. R. (2014). Ferramentas modernas para programação multithread. In Salgado, A. C., Lóscio, B. F., Alchieri, E., and Barreto, P. S., editors, *Jornadas de Atualização em Informática*, pages 41–83. Sociedade Brasileira de Computação, Porto Alegre.

Chu, S.-L., Chen, S.-R., and Weng, S.-F. (2013). CPPM: a comprehensive power-aware processor manager for a multicore system. *Applied Mathematics & Information Sciences*, 7:793–800.

Gautier, T., Roch, J.-L., and Villard, G. (1995). Regular versus irregular problems and algorithms. In *Proc. of IRREGULAR'95, Lyon, France*, pages 1–26. Springer-Verlag.

Goel, B. (2011). Per-core power estimation and power aware scheduling strategies for CMPs. Master's thesis, Institutionen för data- och informationsteknik, Datorteknik (Chalmers), Chalmers tekniska högskola,. 70.

Graham, R. L. (1972). Bounds on multiprocessing anomalies and related packing algorithms. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), pages 205–217, New York, NY, USA. ACM.

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

Houben, C. K. and Halang, W. A. (2014). An energy-aware dynamic scheduling algorithm for hard real-time systems. In *Embedded Computing (MECO), 2014 3rd Mediterranean Conference on*, pages 14–17.

Intel (2013). *Intel 64 and IA-32 Architectures Software Developer's Manual*.

Joseph, R. and Martonosi, M. (2001). Run-time power estimation in high performance microprocessors. In *Low Power Electronics and Design, International Symposium on, 2001.*, pages 135–140, Huntington Beach. ACM.

Jotwani, R., Sundaram, S., Kosonocky, S., Schaefer, A., Andrade, V., Constant, G., Novak, A., and Naffziger, S. (2010). An x86-64 core implemented in 32nm soi cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2010 IEEE International*, pages 106–107.

Libutti, S., Massari, G., Bellasi, P., and Fornaciari, W. (2014). Exploiting performance counters for energy efficient co-scheduling of mixed workloads on multi-core platforms. In *Proceedings of Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, PARMA-DITAM '14, pages 27:27–27:32, New York. ACM.

Lu, Y.-H., Benini, L., and De Micheli, G. (2000). Low-power task scheduling for multiple devices. In *Hardware/Software Codesign, 2000. CODES 2000. Proceedings of the Eighth International Workshop on*, pages 39–43.

McCullough, J. C., Agarwal, Y., Chandrashekar, J., Kuppuswamy, S., Snoeren, A. C., and Gupta, R. K. (2011). Evaluating the effectiveness of model-based power characterization. In *Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'11, Berkeley. USENIX Association.

Nichols, Bradford and Buttlar, Dick and Farrell, Jacqueline (1996). Pthreads programming: a posix standard for better multiprocessing.

OpenMP ARB (2013). Openmp application program interface. Acessado: 07-jul-2015.

Pallipadi, V. and Starikovskiy, A. (2006). The Ondemand Governor: past, present and future. In *Proceedings of Linux Symposium, vol. 2, pp. 223-238*, Ottawa.

Patki, T., Lowenthal, D. K., Rountree, B., Schulz, M., and de Supinski, B. R. (2013). Exploring hardware overprovisioning in power-constrained, high performance computing. In *Proceedings of the 27th International ACM Conference on International Conference on Supercomputing*, ICS '13, pages 173–182, New York, NY, USA. ACM.

Petrucci, V., Loques, O., Mossé, D., Melhem, R., Gazala, N. A., and Gobriel, S. (2015). Energy-efficient thread assignment optimization for heterogeneous multicore systems. *ACM Trans. Embed. Comput. Syst.*, 14(1):15:1–15:26.

Press, W. H., Flannery, B. P., Teukolsky, S. A., and Vetterling, W. T. (1988). *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA.

Rotem, E., Naveh, A., Rajwan, D., Ananthakrishnan, A., and Weissmann, E. (2012). Power-management architecture of the Intel microarchitecture code-named Sandy Bridge. *Micro, IEEE*, 32(2):20–27.

Sheikh, H. F., Tan, H., Ahmad, I., Ranka, S., and Bv, P. (2012). Energy- and performance-aware scheduling of tasks on parallel and distributed systems. *J. Emerg. Technol. Comput. Syst.*, 8(4):32:1–32:37.

Shekar, V. and Izadi, B. (2010). Energy aware scheduling for dag structured applications on heterogeneous and dvs enabled processors. In *Green Computing Conference, 2010 International*, pages 495–502.

Venkatachalam, V. and Franz, M. (2005). Power reduction techniques for microprocessor systems. *ACM Comput. Surv.*, 37(3):195–237.

Walter, B., Bala, K., Kulkarni, M., and Pingali, K. (2008). Fast agglomerative clustering for rendering. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, pages 81–86.

Yang, X., Zhou, Z., Wallace, S., Lan, Z., Tang, W., Coghlan, S., and Papka, M. E. (2013). Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 60:1–60:11, New York. ACM.