Análise de Características Comportamentais de Aplicações OpenMP para Redução do Consumo de Energia

Gabriel B. Moro, Lucas Mello Schnorr

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

{gbmoro,schnorr}@inf.ufrgs.br

Resumo. *Desempenho e consumo energético são requisitos fundamentais em sistemas de computação. Um desafio comumente encontrado é conciliar esses dois aspectos, buscando manter o mesmo desempenho, consumindo cada vez menos energia. Muitas técnicas possibilitam a redução do consumo de energia em aplicações paralelas, mas na maioria das vezes elas envolvem recursos encontrados apenas em processadores modernos ou um conhecimento amplo das características da aplicação e da plataforma alvo. Nesse trabalho propomos uma biblioteca que altera dinamicamente a frequência de processador, utilizando como base de conhecimento uma análise prévia das regiões limitadas pela memória. Os resultados apresentam uma redução de 1,89% no consumo de energia para aplicação Lulesh com cerca de 0,09% de sobrecarga no tempo de execução, quando comparamos nossa técnica com a política de processador Ondemand do Sistema Operacional Linux.*

Abstract. *Performance and energy consumption are fundamental requirements in computer systems. A very frequent challenge is to combine both aspects, searching to keep the high performance computing while consuming less energy. There are a lot of techniques to reduce energy consumption, but in general, they use modern processors resources or they require specific knowledge about application and platform used. In this paper, we propose a library that dynamically changes the processor frequency according to the application's computing behavior, using a previous analysis of its Memory-Bound regions. The results show a reduction of 1,89% in energy consumption for Lulesh application with an increase of 0,09% in runtime when we compare our approach against the governor Ondemand of the Linux Operating System.*

1. Introdução

Aplicações de alto desempenho são de extrema importância para a humanidade. Cada vez mais cientistas de diversas áreas necessitam de poder de computação para processar modelos complexos com o objetivo de avançar suas pesquisas. Além do desempenho dessas aplicações, o consumo de energia é uma preocupação constante em centros de supercomputadores, os quais fornecem máquinas potentes para executar aplicações que muitas vezes levam segundos, horas ou até mesmo dias em alguns casos. Nesse sentido, muitas pesquisas estão sendo realizadas para reduzir o consumo de energia tanto em nível de processador, trabalhando com diferentes configurações de frequência para o processamento (uso de *Dynamic Voltage and Frequency Scaling* - DVFS), como também a nível

de software, utilizando técnicas de caracterização para conhecer melhor o comportamento de aplicações, a fim de realizar otimizações para redução do consumo de energia.

Cada aplicação possui um perfil comportamental de execução, o que está relacionado com o tamanho de entrada, suas estruturas de dados, a quantidade e o tipo de cálculos realizados, entre outros aspectos. Esse perfil pode ser utilizado para agrupar as aplicações de acordo com o componente computacional que limita o desempenho da aplicação. Sendo assim, podemos ter aplicações cujo desempenho é limitado pelos acessos à memória (*Memory-Bound*), outras que são limitadas pelo desempenho computacional do processador (*CPU-Bound*) e enfim aquelas limitadas pelas operações de entrada e saída (*IO-Bound*). [Jesshope and Egan 2006] definem aplicações do tipo *Memory-Bound* como aquelas em que o desempenho aumenta à medida que for reduzido a taxa de *Cache misses* para o segundo nível de *Cache*. Já nas aplicações do tipo *CPU-Bound*, uma redução da taxa de *Cache misses* não necessariamente se transformaria em um melhor desempenho, pois o limitante é a espera pela CPU ou por IO (entrada/saída) e não a memória.

Embora seja possível classificar as aplicações nessas categorias, nem sempre uma aplicação terá um comportamento totalmente *Memory-Bound* ou *CPU-Bound*. O que pode ocorrer é que diferentes regiões de código de uma mesma aplicação podem apresentar uma espera mais marcante por um determinado recurso, seja ele de CPU, Memória ou IO. Aplicações como o *benchmark* Lulesh [Karlin et al. 2013], por exemplo, possuem diferentes limitantes computacionais em suas regiões paralelas. Algumas regiões tem baixa taxa de *misses* na *Cache* L2 e um alto IPC (*Instruction Per Cycle*), da mesma forma, existem outras em que o comportamento é equilibrado entre essas duas medidas (mais detalhes em [Moro and Schnorr 2017]).

A partir de uma caracterização comportamental das regiões *Memory-Bound* de uma aplicação, seria possível escolher regiões potenciais a serem executadas em uma frequência menor, com o objetivo de reduzir o consumo de energia. Nesse cenário, o principal desafio é a coleta de informações detalhadas durante a execução da aplicação, as quais possam fornecer as características de cada região da aplicação paralela.

O objetivo desse trabalho é propor uma metodologia de análise de desempenho que permita detectar o comportamento de regiões paralelas em aplicações OpenMP, através do uso de medidas como IPC (*Instruction Per Cycle*) e taxa de *Misses* na *Cache* L2. É proposto nesse trabalho um *Workflow* de análise para identificar quais regiões são *Memory-Bound*, a partir dessa investigação, o *Workflow* atua realizando um controle dinâmico de frequência de acordo com o comportamento da aplicação.

O presente artigo está organizado da seguinte maneira. Na Seção 2 é abordado sobre os tipos de aplicações paralelas, suas características e os principais conceitos de consumo de energia. A Seção 3 é apresentado o *Workflow* de investigação de regiões *Memory-Bound*, o qual é definido em duas fases, a primeira de investigação das regiões e a segunda que realiza o controle dinâmico de frequência. A Seção 4 apresenta os resultados do experimento em que comparamos a política *Ondemand* com a biblioteca proposta. A Seção 5 apresenta uma comparação de nosso trabalhos com o estado da arte. A Seção 6 encerra o artigo com as considerações finais e trabalhos futuros.

2. Comportamento de Aplicações Paralelas e Consumo de Energia

Segundo Diamond et al. [Diamond et al. 2011] existem três padrões de aplicação em computação de alto desempenho: as aplicações regulares que possuem computações e acessos à memória, os quais são previsíveis; as irregulares que possuem acessos aleatórios a memória, como também conseguem alterar dinâmicamente suas estruturas de dados; e as aplicações de grafos que realizam acessos em diferentes localidades da memória, fazendo com que a localidade espacial da *Cache* seja menor. Nessas diferentes classes de aplicações é possível encontrar regiões paralelas que possuem uma espera maior pela memória e outras que são mais limitadas pela CPU.

Ao realizar a leitura de uma determinada informação, a aplicação realiza buscas nas memórias *Cache*. Não encontrando a informação, é necessário acessar a memória principal. O acesso à memória principal gera um custo maior de desempenho, visto que, a memória principal possui uma velocidade inferior a da memória *Cache*, pois são tecnologias diferentes para fins diferentes. Nisso, a taxa de acertos dessa busca interfere diretamente no desempenho de aplicações *Memory-Bound*, pois quanto menos vezes for necessário acessar a memória principal, maior será o desempenho da aplicação. Aplicações consideradas como *Memory-Bound* possuem uma grande parte de sua execução ociosa, interferindo no desempenho da aplicação como um todo [Subramanian 2015]. Geralmente esse comportamento é encontrado em aplicações SIMD (*Single Instruction Multiple Data*), tais como processamento de imagens, vídeos e assim por diante.

Diferente das aplicações *Memory-Bound*, as aplicações *CPU-Bound* são limitadas pela espera por CPU. Essas aplicações são aquelas que possuem o desempenho totalmente afetado pela velocidade em que o processador realiza as operações aritméticas solicitadas [Hutcheson and Natoli 2011]. Um bom exemplo deste tipo de aplicação é a sequência de Fibonacci, a qual solicita à CPU, a computação de várias instruções por ciclo, devido sua recursividade e demais cálculos realizados.

Segundo [Orgerie et al. 2014], o consumo de energia de recursos computacionais pode ser medido através de sensores de energia (gasto real) ou estimados a partir de modelos (estimativa teórica). Algumas ferramentas podem ser utilizadas para acessar esses sensores. Um exemplo de ferramenta que permite obter uma relação do consumo de energia total gasto pela aplicação é a Intel PCM, fornecendo o consumo de energia gasto pela CPU (considerando as memórias *Cache*) e memória principal [Silveira et al. 2016]. Outra maneira é estimar a energia utilizando modelos, o CACTI é um exemplo de ferramenta que permite especificar o modelo de memória utilizado pela plataforma de execução, a partir disso é possível obter uma estimativa de gasto de energia por instrução [Silveira et al. 2016].

Além de conhecer o quanto de energia a aplicação consome, vários trabalhos possuem propostas de redução no consumo energético, dentre os assuntos investigados, uma técnica amplamente utilizada para economizar energia a partir da redução da frequência de processador é a técnica DVFS (*Dynamic Voltage and Frequency Scaling*). Essa técnica permite reduzir a frequência de processador em determinados trechos de execução do programa, os quais possuem um comportamento mais *Memory-Bound* [Sueur and Heiser 2010].

Dynamic Voltage and Frequency Scaling (DVFS) é uma técnica muito utilizada em sistemas embarcados, *desktops* e também em *workstations* que executam aplicações de alto desempenho [Sueur and Heiser 2010]. Geralmente, essa técnica é utilizada em

conjunto com metodologias que definem o momento exato para a troca de frequência ocorrer. Dependendo do tipo de aplicação e da metodologia empregada, a técnica DVFS pode ser vantajosa, tanto em termos de consumo de energia, quanto em desempenho. O *driver* CPUFreq é um exemplo de utilização da técnica DVFS no Sistema Operacional Linux, ele fornece uma interface para alterar a frequência dos processadores e utilizar políticas pré-implementadas [Brodowski et al. 2013]. O *framework* disponibiliza as seguintes políticas (modos): *Performance* - utiliza a maior frequência disponível; *Powersave* - utiliza a mais baixa frequência; *Ondemand* - escolhe a frequência de acordo com o uso da CPU; *Conservative* - funciona como o *Ondemand*, mas o ajuste é mais refinado; e *Userspace* - utiliza a frequência informada pelo usuário ou por algum programa executado como super usuário [Brodowski et al. 2013].

3. Workflow de Investigação de Regiões Memory-Bound

A Figura 1 apresenta uma visão geral do *Workflow* proposto. O programa paralelo escrito em OpenMP é executado com uma biblioteca externa que, permite a observação e o controle de frequências do processador. Essa biblioteca com dois comportamentos permite a execução do *Workflow* em duas fases.

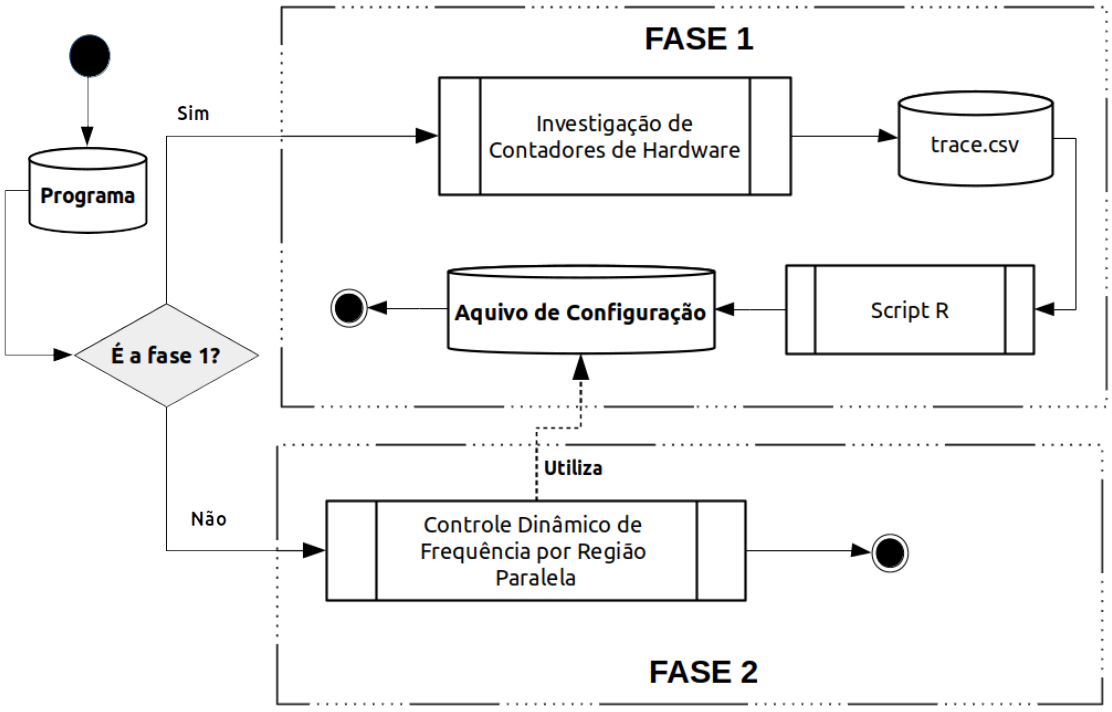

Figura 1. Visão Geral do *Workflow* de investigação e controle de frequência, com duas fases: (1) contadores de hardware são coletados e os dados interpretados a partir de scripts em R que permitem gerar um arquivo de configuração para a fase seguinte; (2) baseado em um arquivo de configuração que associa regiões paralelas a frequências de processador específicas o programa é executado com o objetivo de reduzir o consumo de energia.

Na primeira fase (parte superior na figura), o programa paralelo em OpenMP, durante sua execução, terá seu comportamento registrado. Esse comportamento contém nos

registros os contadores de hardware relevantes para identificar regiões de código limitadas pelos acessos à memória. Como resultado intermediário da primeira fase, obtém-se um arquivo de rastro que pode ser potencialmente grande dependendo da quantidade de regiões paralelas, da duração do programa e do tamanho do problema. Os rastros tem, em cada evento, o identificador da região paralela OpenMP (nome do arquivo onde aparece e número da linha), os contadores de hardware, o início e a duração da região paralela. Caso uma região paralela seja executada várias vezes, teremos no rastro várias ocorrências do seu comportamento.

Ainda na primeira fase, esse rastro é processado através de um *script* R que, faz uma análise de dados para identificar quais regiões paralelas tem características do estilo *Memory-Bound*. Como conclusão da primeira fase, obtemos um arquivo de configuração que será realimentado para a biblioteca externa que, muda seu comportamento para a fase seguinte. Esse arquivo de configuração tem um mapeamento das regiões paralelas para a frequência do processador apropriada, mediante edição manual pelo analista. Após isso na segunda fase, e de posse do arquivo de configuração, a biblioteca externa, ao invés de coletar informações, altera a frequência do processador durante a execução do programa.

O resto desta seção apresenta maiores detalhes de cada uma das fases assim como um detalhamento da implementação da biblioteca `LibNina` que implementa este *workflow* e permite a observação/controle de programas paralelos escritos em OpenMP.

3.1. Fase 1: Obtenção da Assinatura Computacional

A partir de contadores de hardware é possível obter medidas como o total de ciclos de CPU, acessos à *Cache* ou memória principal, quantidade de instruções de tipo *load*, entre outras. Como em nossa abordagem desejamos conhecer quais são as regiões da aplicação que são limitadas pela memória, utilizamos contadores de hardware para obter o total de instruções executadas, quantidade de ciclos de CPU, total de acessos e *misses* na *Cache* L2.

De forma geral, ao executar aplicações OpenMP, o processo principal dispara várias *threads* de processamento. Portanto, precisamos conhecer o comportamento das regiões executadas por cada *thread*, desprezando a parte sequencial do programa. Na primeira fase, os contadores de hardware são coletados por *thread*, a fim de que no final, agrupando todos os contadores por *thread*, seja possível identificar as regiões mais limitadas pela memória. O IPC (*Instruction per Cycle*) permite mapear quantas instruções são realizadas por ciclo de processador, essa porcentagem é extremamente importante para reconhecer em aplicações paralelas, o comportamento *CPU-Bound*. Já a taxa de *misses* na *Cache* L2 permite mapear a porcentagem de insucesso nas requisições realizadas pela aplicação à memória.

Na execução da primeira fase não é necessário executar a aplicação com uma entrada grande, pois quanto maior a entrada, mais tempo a aplicação levará para processar e maior será o rastro a ser analisado. A partir disso, o resultado da primeira fase é um arquivo de configuração para a biblioteca. Ele é obtido por análise manual, utilizando como entrada o rastro de comportamento. As frequências de processador que deverão ser utilizadas serão definidas para cada região paralela. A partir dos valores dos contadores de hardware, estabelecemos uma relação entre as duas métricas (IPC e taxa de *Misses* na *Cache* L2). As regiões identificadas como *Memory-Bound*, por exemplo, podem receber

uma frequência de processador menor do que as demais.

3.2. Fase 2: Controle Dinâmico da Frequência por Região Paralela

O controle dinâmico da frequência é realizado de acordo com o arquivo de configuração gerado na fase anterior. As frequências a serem configuradas são armazenadas em uma estrutura de dados, ao entrar em uma região é buscado nessa estrutura de dados, a frequência de processador associada a sua respectiva região. Como a troca de frequência é custosa em termos de desempenho (devido a latência da troca da frequência uma vez que o comando é dado pelo Sistema Operacional), realizamos ela apenas quando necessário, levando em consideração a duração da região paralela. A troca de frequência ocorre sempre ao entrar na região paralela, ao sair não é realizado nenhuma configuração de frequência, visto que ao entrar em uma nova região, uma nova frequência será atribuída para a mesma.

3.3. Implementação do workflow na `LibNina`: DVFS para Regiões Paralelas

Uma biblioteca de código aberto[1] chamada `LibNina` foi desenvolvida para implementar o *Workflow* em duas fases. A aplicação paralela com regiões OpenMP é instrumentada automaticamente com um compilador fonte-para-fonte (*source-to-source*) chamado `Opari2`. A instrumentação consiste em chamadas a funções específicas que, são implementadas pela `LibNina` de forma que, elas possam observar/controlar a execução da aplicação OpenMP. Dentre os múltiplos pontos de instrumentação possíveis, a `LibNina` adota especificamente os pontos de código onde um *Parallel_begin* e um *Parallel_end* são executados. Eles representam o início e o fim de blocos de código marcados por `#pragma parallel` da especificação OpenMP.

No momento que cada região paralela começa e termina de ser executada, nos pontos de instrumentação descritos acima, funções específicas da `LibNina` são chamadas para atuarem conforme o *Workflow*. O mapeamento das regiões paralelas é possível tendo em vista que o compilador `Opari2` instrumenta o código criando identificadores únicos para cada região paralela, contendo o nome do arquivo, e a linha de código de início e fim de cada região paralela. A LibNina contém dois componentes principais, o primeiro destinado a utilizar a biblioteca `PAPI` para coletar os contadores de hardware (na primeira fase do *Workflow*), o segundo destinado a alterar a frequência do processador utilizando a biblioteca `CPUFreq` (na segunda *fase*).

4. Resultados

A aplicação utilizada nos experimentos é o *benchmark* Lulesh [Karlin et al. 2013], o qual representa uma simulação computacional de modelagem hidrodinâmica. Essa simulação reproduz uma onda de explosão em um espaço confinado, utilizando como base o método de Sedov [LLNL 2018]. A aplicação é reconhecida por ter características *Memory-Bound*. Por essa razão, ela foi utilizada pela perspectiva de ganhos potenciais, ao executar tais regiões em uma frequência de processador mais baixa. Para fins de comparação, adotamos o gerenciador de frequência *Ondemand* do pacote CPUFreq do Sistema Operacional Linux que, realiza a troca de frequência tendo como base a taxa de uso do processador, sem qualquer relação com a aplicação executada.

[1]Código disponível em `https://github.com/tido4410/libnina`

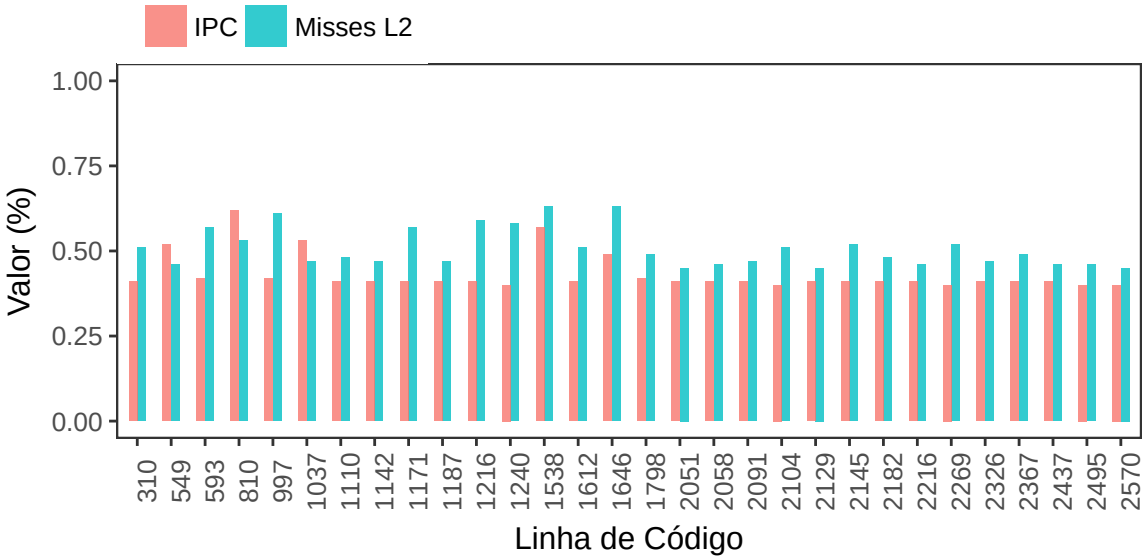

Figura 2. Porcentagem de IPC e Misses na Cache L2 por Linha de Código.

A plataforma utilizada para executar os experimentos é uma *Workstation* de 126G de memória RAM e dois processadores Intel(R) Xeon(R) CPU E5-2650, cada um com 10 núcleos físicos que possuem tecnologia *HyperThreading*. A máquina permite a frequência máxima de 2.30GHz e mínima de 1.2GHz. O Sistema Operacional da plataforma é o Ubuntu 16.04LTS com o framework CPUFreq instalado e suporte as políticas *Ondemand*, *Consertative*, *Performance*, *Powersave* e *Userspace*. Todas as combinações de configuração para os experimentos foram executadas utilizando 20 threads fixas nos cores físicos, com a ausência do uso de *HyperThreading* que foi desativado explicitamente. Para lidar com a variabilidade natural do sistema computacional, os experimentos foram replicados e médias aritméticas simples são calculadas e relatadas. Adotamos a ferramenta AppPowerMeter[2] para realizar as medições de consumo de energia.

A primeira fase do *Workflow* é executar a aplicação Lulesh com uma entrada menor, a fim de avaliar o comportamento de suas regiões paralelas, investigando alguns contadores de hardware. A execução utiliza apenas 10 elementos de entrada, dessa forma é possível obter um tamanho de rastro adequado para análise. No experimento utilizamos quatro contadores: *PAPI_L2_TCA*, *PAPI_L2_TCM*, *PAPI_TOT_INS* e *PAPI_TOT_CYC*. Esses contadores permitem obter as métricas IPC (*Instruction per Cycle*) e a taxa de *Misses* na *Cache* L2. O resultado da primeira fase pode ser visto na Figura 2 que, apresenta a investigação das duas medidas por região paralela do código da aplicação Lulesh. Adotamos como identificador da região paralela o número da linha que começa aquela região (o local do `#pragma omp parallel`), sendo utilizada portanto no eixo horizontal da figura (o código é contido em um único grande arquivo fonte).

Na Figura 2 é possível visualizar a relação entre as duas medidas obtidas: o IPC e a taxa de *Misses* na *Cache* L2. O eixo "x" apresenta o identificador (Linha de Código) na região paralela onde a medida é obtida. A coleta dos contadores de hardware é realizada em dois momentos, ao entrar e sair de uma região, com isso é possível obter as medidas

[2]Disponível em https://github.com/kentcz/rapl-tools

de IPC e taxa de *Misses* na *Cache* L2 para cada região. Com essa investigação, adotamos várias estratégias diferentes para economia de energia. Cada estratégia se reflete em um arquivo de configuração diferente para a segunda fase de nosso *Workflow*. Dentre as estratégias, as mais proeminentes são: `(a)` o uso de frequência máxima nas regiões onde o IPC é maior do que 50%, sendo que nas demais localizações o uso de frequência mínima; `(b)` o uso de frequência máxima em todas localizações exceto em regiões onde a taxa de misses é maior do que 50% (nessas regiões o uso de frequência mínima); `(c)` e a combinação das duas configurações anteriores, uso de frequência baixa quando o IPC é menor do que 50% e a taxa de misses é maior do que 50%, nas demais regiões o uso de frequência máxima. Essas configurações possibilitaram identificar uma sobrecarga na troca de frequência, presente na implementação da biblioteca `LibNina`. A razão desta sobrecarga tem origem na própria aplicação Lulesh que, possui regiões cuja assinatura computacional é bastante diferente (Figura 2). Isso obriga a `LibNina` a adaptar a frequência do processador em mais ocasiões em um intervalo de tempo curto, gerando um custo extra no consumo de energia. Com isso, nós identificamos que, em regiões de curta duração (tempo), caso seu comportamento solicitasse uma troca de frequência, seria mais vantajoso executar tal região sem a troca de frequência, a fim de evitarmos o sobrecusto. Os melhores resultados foram obtidos com esta estratégia adaptada que combina a assinatura computacional (contadores de hardware) com a duração das regiões paralelas.

A Figura 3 apresenta o consumo de energia (em X) em função da política de gerenciamento da frequência do Linux *Ondemand* e da biblioteca proposta (em Y), para dois tamanhos de entrada: 160 e 180. Os dados do gráfico foram obtidos a partir de uma bateria experimental que inclui cinco replicações de cada configuração experimental totalizando vinte execuções que são executadas em ordem aleatória, cada uma com uma duração média de 1 hora. A partir dos resultados, é possível verificar que, ao utilizar 160 elementos, o uso da biblioteca reduziu cerca de aproximadamente 1.44% o consumo de energia, comparado com o modo *Ondemand*. Já utilizando 180 elementos, ao utilizar a biblioteca, o consumo de energia reduziu em cerca de 1.89%. Embora pequenos, os ganhos são consistentes tendo em vista a longa duração (≈1 hora) dos experimentos.

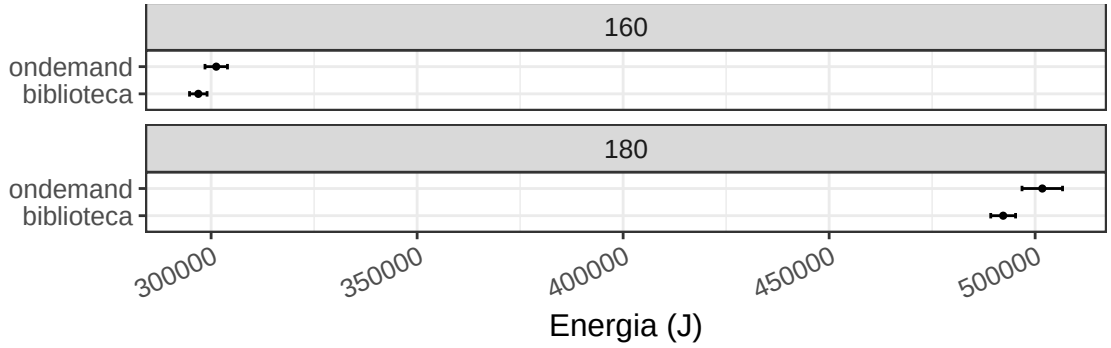

Figura 3. Lulesh - Consumo de Energia - Governor Ondemand vs Biblioteca.

A Figura 4 apresenta o consumo energético (em Y) correlacionado com o tempo de execução (em X) da aplicação Lulesh quando esta é executada com o tamanho da entrada de 180 em dois cenários: o caso *Ondemand*, utilizado como referência, e a biblioteca (nossa abordagem). O uso da biblioteca possibilita um ganho no consumo de energia, conforme visto anteriormente, mas em termos de tempo de execução, o modo *Ondemand*

é aproximadamente 0,09% mais rápido. No gráfico 4 é possível visualizar essa relação entre o consumo de energia, como também a perda correspondente no tempo de execução, da biblioteca sobre a política *Ondemand*.

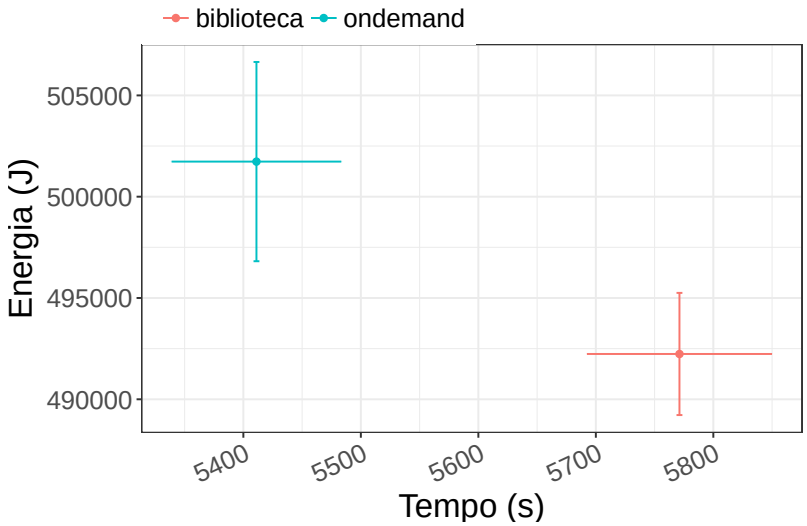

Figura 4. Lulesh - Tamanho 180 - Tempo e Energia.

5. Trabalhos Relacionados

Molka et al. [Molka et al. 2017] apresentam uma abordagem que investiga quais contadores de hardware devem ser utilizados para se obter uma melhor estimativa do uso de componentes da hierarquia de memória. Para isso, os autores utilizam um conjunto de *benchmarks* específicos, os quais tem por objetivo o "estresse" individual dos componentes do sistema de memória (*Caches* e Memória Principal). Os *benchmarks* utilizados no trabalho permitem configurações como: restrição para medir apenas uma única CPU, configuração de uso de apenas 50% da *Cache* L1, diminuição do tamanho do último nível de *Cache* para menos de 10 vezes, dimensionamento de *loads* e *stores*, entre outras configurações para melhor examinar os componentes de memória utilizando apenas um processador, como também recursos compartilhados no cenário de uma plataforma *multi-core*. Os ciclos de processador são divididos primeiramente em dois grupos, os ciclos de produtividade que são aqueles onde ocorre o processamento útil de instruções e os ciclos de espera que existem em funçao da latência gerada pelo acesso à mémória ou por outras razões. Nos ciclos *Memory-Bound*, os autores identificaram duas situações, a primeira quando os ciclos de processador eram limitados pela espera por memória e a segunda situação quando os ciclos eram limitados pela largura de banda da memória. Os autores mapearam os ciclos ativos e os ciclos de espera em uma aplicação OpenMP do *benchmark* NAS, após isso é realizado uma redução de frequência de processador nos momentos onde a aplicação possuia ciclos de espera, através disso, os autores reduzem o consumo de energia em cerca de 4,16% com uma penalidade de 0,2% no tempo de execução.

Millani e Schnorr [Millani and Schnorr 2016] investigam a relação entre tempo e energia, ao adotar frequências de processador por região paralela. Primeiramente, os autores adotam um projeto experimental do tipo *screening* para identificar as regiões paralelas mais propícias para redução do consumo energético, sem penalidade em tempo de

execução. Nesta fase, as regiões paralelas são assinaladas com duas possíveis frequências (baixa ou alta), as quais devem ser suportadas pela plataforma alvo. Os autores utilizam as técnicas de Análise de Variância (ANOVA) e análise de efeitos principais (*Main Effects Analysis*) para identificar as regiões paralelas potenciais, aquelas que possuem maior impacto no consumo de energia e menor impacto no tempo de execução. Dessa forma, em uma nova execução do programa, apenas as regiões específicas (potenciais) sofrerão a alteração de frequência de processador. Dentre os resultados obtidos, os autores conseguiram uma redução de aproximadamente 9,27% no consumo de energia para o *benchmark* MiniFE, sem perdas significativas de desempenho. Por outro lado, a abordagem é totalmente dependente de instrumentação de código, exigindo um conhecimento prévio da aplicação ao atribuir a frequência de processador para as regiões paralelas, sendo uma desvantagem em cenários onde a aplicação é completamente desconhecida pelo analista de desempenho.

Existem trabalhos como o de Wang et al. [Wang et al. 2015] que adotam uma abordagem voltada mais às técnicas de gerenciamento de potência, as quais são disponibilizadas pelo próprio hardware. Ao invés de utilizar DVFS para alternar a frequência de processador a ser utilizada, os autores utilizam CPU *Clock Modulation* para definir a frequência adequada para os diferentes *loops* da aplicação. Dessa forma, é possível visualizar ganhos ao utilizar execuções com múltiplas frequências que são selecionadas com CPU *Clock Modulation*. Ao entrar em cada laço do programa, a frequência de processador é reduzida e ao sair a mesma é aumentada. Segundo os autores, a técnica pode em alguns casos acarretar um aumento no tempo de execução, mas um ganho considerável no consumo de energia. O melhor resultado apresentado pelos autores é de uma redução média de 8,6% no consumo de energia, com a penalidade inferior a 1,5% no tempo de execução. Por utilizar CPU *Clock Modulation*, a metodologia só pode ser aplicada em plataformas com processadores recentes.

Laurenzano et al. [Laurenzano et al. 2011] apresenta uma técnica que permite selecionar frequências de processador adequadas por laço de repeticação da aplicação. Na caracterização das aplicações, os autores utilizam a taxa de *hit* na *Cache* L1, L2 e L3; a taxa de operações com ponto flutuante para o número de operações de memória; e a taxa de operações com números inteiros por operações com ponto flutuante. Os autores formam uma base de conhecimento de acordo com inúmeras execuções, caracterizando o consumo de energia gasto de acordo com padrões de execução e frequências de processador executadas anteriormente, obtidas por rastro. Dessa maneira é possível buscar nesse conjunto de cobertura, qual a melhor frequência para determinado padrão de aplicação. O melhor caso apresentado pelos autores é uma redução de aproximadamente 10,6% no consumo de energia, utilizando determinada carga de trabalho para uma arquitetura de 32 cores AMD Opteron 8380. Existem basicamente duas diferenças com nosso trabalho: (1) deve ser construída uma base de conhecimento baseada em código artificial; (2) a base deve ser consultada em tempo de execução para selecionar a frequência de processador mais apropriada, utilizando métricas de desempenho coletadas dinamicamente.

Em contrapartida, Shafik et al. [Shafik et al. 2015] apresentam uma abordagem para redução do consumo de energia utilizando políticas de hardware, tais como DVFS e *Dynamic Concurrency Throtling* (DCT). A abordagem investiga o desempenho da parte sequencial e paralela do programa, gerando anotações que serão utilizadas como uma

espécie de marcadores, os quais descrevem o desempenho do programa. Esses marcadores são posteriormente incorporados a biblioteca OpenMP, como modificações na libgomp. Através de estatísticas obtidas por contadores de hardware em intervalos regulares são aplicadas as técnicas DVFS e DCT, tanto na parte sequencial como na parte paralela da aplicação. Dentre os resultados, os autores apresentam uma redução de aproximadamente 17% no consumo de energia. Como a técnica proposta por [Wang et al. 2015], a abordagem de [Shafik et al. 2015] depende de recursos de hardware, os quais podem ser encontrados apenas em processadores mais recentes.

Dentre os trabalhos analisados é perceptível que existem várias formas de caracterizar uma aplicação paralela, sendo pelo uso de contadores de hardware, medidas de tempo de computação/comunicação, intervalos de tempo específicos em que a CPU fica ociosa e assim por diante. Algumas abordagens utilizam recursos disponibilizados pelo próprio hardware, o que possibilita melhores resultados na maioria dos casos. Vale salientar que, a maior parte das abordagens que reduzem o consumo de energia acarretam sobrecargas no tempo de execução. Na maioria dos casos é possível observar que, os trabalhos que utilizam técnicas fornecidas pelo próprio hardware resultam em maiores ganhos de energia. Além do que, se limitam a plataformas específicas ou processadores recentes. Nesse contexto, o presente trabalho propõe uma abordagem em formato de *Workflow* que pode ser aplicada em qualquer plataforma que possua suporte a política *Userspace* do *driver* CPUFreq. A abordagem proposta não requer nenhum conhecimento prévio sobre a aplicação alvo, assim o analista de desempenho pode utilizá-la em qualquer aplicação OpenMP.

6. Considerações Finais

O principal objetivo desse trabalho foi o de projetar e implementar um *Workflow* de investigação de regiões paralelas em aplicações OpenMP. Organizada em duas fases, a partir dos contadores de hardware (fase 1), a metodologia propõem a alteração da frequência de execução das regiões (fase 2) para uma configuração adequada a sua assinatura computacional: caso a região paralela seja limitada pela memória, reduz-se a frequência de execução do processador naquele trecho. A abordagem foi implementada na biblioteca LibNina, de código aberto, a qual utiliza informações a respeito do comportamento da aplicação fornecida pela primeira fase e depois altera a frequência de processador do programa de acordo com o comportamento respectivo de cada região paralela, alterando entre frequência máxima e mínima. Os resultados obtidos com a aplicação Lulesh indicam um ganho de 1,89% no consumo de energia sobre a política *Ondemand* do Linux, utilizada como referência. Vale salientar que, ocorreu um aumento no tempo de execução, cerca de 0,09%, no pior caso testado. Acreditamos que esse aumento no tempo de execução esteja relacionado a sobrecarga imposta pela implementação da biblioteca e aos frequentes comandos de adaptação da frequência do processador.

Como trabalhos futuros, prentendemos investigar alternativas para reduzir o custo da troca de frequência em regiões paralelas de curta duração, as quais possuem uma duração temporal igual ou menor ao tempo gasto pela operação de troca de frequência. Nesse cenário, a sobrecarga da operação de troca de frequência não se justificaria e poderia ser amenizada. Também, pretendemos analisar outros contadores de hardware, como por exemplo, quantidade de *loads* e *stores* realizados pela aplicação em cada região paralela. Como a `LibNina` suporta a configuração de quais contadores de hardware utilizar,

essa investigação seria facilitada pela biblioteca.

Agradecimentos

Nós agradecemos os projetos FAPERGS/Inria ExaSE, FAPERGS 16/2551-0000 354-8 (PPP 2014), FAPERGS 16/2551-0000 488-9 (PRONEX 2014), CNPq 447311/2014-0, e o programa H2020 da União Europeia e o MCTI/RNP através do projeto HPC4E, identificador 689772.

Referências

[Brodowski et al. 2013] Brodowski, D., Golde, N., Wysocki, R. J., and Kumar, V. (2013). Linux cpufreq governors – information for users and developers. Technical report, Linux.

[Diamond et al. 2011] Diamond, J., Burtscher, M., McCalpin, J. D., Kim, B.-D., Keckler, S. W., and Browne, J. C. (2011). Evaluation and optimization of multicore performance bottlenecks in supercomputing applications. In *Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium on*, pages 32–43. IEEE.

[Hutcheson and Natoli 2011] Hutcheson, A. and Natoli, V. (2011). Memory bound vs. compute bound: A quantitative study of cache and memory bandwidth in high performance applications. Technical report, Stone Ridge Technology.

[Jesshope and Egan 2006] Jesshope, C. and Egan, C. (2006). *Advances in Computer Systems Architecture: 11th Asia-Pacific Conference, ACSAC 2006, Shanghai, China, September 6-8, 2006, Proceedings*. LNCS sublibrary: Theoretical computer science and general issues. Springer.

[Karlin et al. 2013] Karlin, I., Keasler, J., and Neely, R. (2013). Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, LLNL.

[Laurenzano et al. 2011] Laurenzano, M. A., Meswani, M., Carrington, L., Snavely, A., Tikir, M. M., and Poole, S. (2011). Reducing energy usage with memory and computation-aware dynamic frequency scaling. In *European Conference on Parallel Processing*, pages 79–90. Springer.

[LLNL 2018] LLNL (2018). Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254, LLNL.

[Millani and Schnorr 2016] Millani, L. F. and Schnorr, L. M. (2016). Computation-aware dynamic frequency scaling: Parsimonious evaluation of the time-energy trade-off using design of experiments. In *European Conference on Parallel Processing*, pages 583–595. Springer.

[Molka et al. 2017] Molka, D., Schöne, R., Hackenberg, D., and Nagel, W. E. (2017). Detecting memory-boundedness with hardware performance counters. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering*, pages 27–38. ACM.

[Moro and Schnorr 2017] Moro, G. B. and Schnorr, L. M. (2017). Detecting memory-bound parallel regions to improve the energy-efficiency of applications. In *XIV Workshop de Processamento Paralelo e Distribuído*, pages 10–13.

[Orgerie et al. 2014] Orgerie, A.-C., de Assuncao, M. D., and Lefevre, L. (2014). A Survey on Techniques for Improving the Energy Efficiency of Large-scale Distributed Systems. *ACM Comput. Surv.*, 46(4):47:1–47:31.

[Shafik et al. 2015] Shafik, R. A., Das, A., Yang, S., Merrett, G., and Al-Hashimi, B. M. (2015). Adaptive energy minimization of openmp parallel applications on many-core systems. In *Proceedings of the 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures*, pages 19–24. ACM.

[Silveira et al. 2016] Silveira, D. S., Moro, G. B., Cruz, E. H. M., Navaux, P. O. A., Schnorr, L. M., and Bampi, S. (2016). Energy Consumption Estimation in Parallel Applications: an Analysis in Real and Theoretical Models. In *WSCAD 2016 - XVII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 134–145.

[Subramanian 2015] Subramanian, L. (2015). Providing high and controllable performance in multicore systems through shared resource management. *arXiv preprint arXiv:1508.03087*.

[Sueur and Heiser 2010] Sueur, E. L. and Heiser, G. (2010). Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8.

[Wang et al. 2015] Wang, W., Porterfield, A., Cavazos, J., and Bhalachandra, S. (2015). Using per-loop cpu clock modulation for energy efficiency in openmp applications. In *Parallel Processing (ICPP), 2015 44th International Conference on*, pages 629–638. IEEE.