ESCALONAMENTO DE *THREADS* ANAHY FOCADO NA REDUÇÃO DO CONSUMO DE ENERGIA DE PROGRAMAS MULTITHREAD EM ARQUITETURAS MULTICORE

ARAUJO, Alan Schlindvein[1]; CAVALHEIRO, Gerson Geraldo Homrich[1]

[1]Universidade Federal de Pelotas - UFPel. Centro de Desenvolvimento Tecnológico.
Caixa Postal 354 – 96010-900 – Pelotas – RS – Brasil.

1 INTRODUÇÃO

A busca por eficiência energética de sistemas computacionais tornou-se um amplo campo de pesquisa nos últimos anos devido a crescente necessidade do uso racional de recursos energéticos. As arquiteturas multicore representam resultados palpáveis destas pesquisas no contexto do processamento paralelo. No entanto, por possuírem múltiplas unidades de processamento e serem muito utilizadas em aplicações de grande custo computacional, estas arquiteturas ainda consomem muita energia para manter-se em operação em sua capacidade máxima.

Além de econômicas em termos de consumo de energia, outro atrativo destas arquiteturas é seu baixo custo (WOLF, 2004) (HILL e MARTY, 2008), motivo pelo qual esse tipo de configuração tornou-se praticamente onipresente a todos os de sistemas computacionais, desde aqueles voltados a usos pessoais até aqueles implantados em centros dedicados ao processamento de alto desempenho. Outra razão de sua popularização pode ser atribuída ao bom desempenho proporcionado por tais arquiteturas em relação ao seu baixo consumo energético, principalmente quando comparados às últimas gerações de arquiteturas monoprocessadas. No entanto, à medida que o número de processadores/*cores* aumenta, torna-se difícil perceber melhorias no desempenho, pois muitas aplicações existentes não são devidamente adaptadas para explorar o paralelismo de *threads* sobre muitos núcleos de processamento. Na realidade muitas aplicações não exploram, durante a execução do programa, todos os recursos que a arquitetura oferece. Ao contrário, existe uma grande classe de aplicações que possuem fluxos de dependência de dados que determinam quais atividades podem ser executadas ou não de forma concorrente. Tais aplicações podem ser descritas sob a forma de um grafo de fluxo de dados, do qual pode ser extraído o caminho crítico, ou seja, a maior sequência de dependência de dados da aplicação.

Neste trabalho é avaliado o impacto no consumo de energia de uma estratégia de escalonamento de *threads* em nível aplicativo. Essa estratégia considera o custo computacional de cada *thread* e gerencia, por meio de seleção de afinidade e controle de frequência, as operações executadas em cada núcleo de processamento do sistema com vistas a minimizar o consumo de energia sem afetar significativamente o desempenho, o que pode ser comprovado nos experimentos.

2 AJUSTANDO A FREQUÊNCIA DE OPERAÇÃO

A partir de experimentos em (WECHSLER, 2006) é possível reduzir o consumo de energia do processador com a redução de sua frequência de operação. No entanto, esta prática resulta em perda de desempenho do processador, uma vez que este passa a operar em velocidade reduzida, implicando em uma inevitável degradação no tempo de execução do programa. No caso de processadores

multicore, executando programas multithread, o raciocínio é o mesmo, para o suporte à execução de *threads* por *cores*. Gerenciar a velocidade de operação de cada *core* permite, portanto, controlar a relação entre consumo de energia e eficiência de execução. Desta forma é possível realizar um escalonamento capaz de distribuir *threads* entre os *cores* adequando velocidade de processamento a partir das cargas de trabalho dos fluxos de execução de um programa. A realização destas operações possui um custo associado, podendo comprometer o desempenho do sistema em geral para a alteração da frequência de operação. Primeiramente, a tensão do processador precisa ser alterada, para que essa possa suportar a frequência exigida, somente após essa etapa a frequência pode ser ajustada para a desejada. Consequentemente, o processador deve interromper suas atividades aguardando que sua frequência seja modificada, ou seja, nessa operação é perdido tempo de processamento (UHRIG e UNGERER, 2005), portanto a heurística implementada para o uso desse recurso dever ser muito bem formulada para que as alterações de frequência de operação não sejam frequentes e assim não tomem tempo de processamento dos *cores*.

3 INTERFACE DE PROGRAMAÇÃO DE ANAHY

O modelo da interface de programação de Anahy (CAVALHEIRO et al., 2007) busca dissociar a concorrência descrita por uma aplicação paralela na arquitetura disponível, com isso o programador pode descrever seu programa paralelo sem considerar os recursos de processamento disponíveis, de modo que o número de atividades concorrentes, ou seja, um número *p* de processadores virtuais (PVs), sempre seja maior que as unidade processamento da arquitetura real. O escalonamento realizado por Anahy ocorre em dois níveis, o primeiro nível é executado a partir de primitivas para o controle de afinidade fornecidas pelo sistema operacional nativo da máquina hospedeira ao conjunto *m* de processadores físicos, tal que $p > m$. O segundo acontece em nível aplicativo, sendo responsável pela alocação das tarefas ao PVs. Nesse nível é considerada a ordem de execução das tarefas, *threads* Anahy uma vez que associados a um PV não sofrem migração, isso reflete o controle semântico do programa, permitindo estimar a carga computacional associada a cada PV e por sua vez determinar o custo de execução do programa sobre as unidades de processamento.

4 RESULTADOS

Para os estudos de caso foram desenvolvidos dois conjuntos de testes e executados em uma máquina Intel® Core™ 2 Quad, 2.66 Ghz, com 64 KB no primeiro nível de cache, 4 MB compartilhados no segundo nível e 4 GB de memória RAM. O primeiro conjunto é chamado ExSF e realiza o cálculo do Fibonacci de forma recursiva. O Segundo algoritmo chamado ExMS realiza a ordenação de uma lista contendo $n * 104$ números usando o algoritmo do MergeSort. Uma característica importante a respeito da implementação dos programas é a introdução de uma carga computacional sintática em cada nodo do grafo, permitindo assim ressaltar o impacto do caminho crítico no programa.

O conjunto de teste ExSF apresenta uma estrutura interessante em relação ao seu grau de paralelismo. Essa estrutura caracteriza a evolução do programa por divisão e conquista, onde são expandidos, tanto quanto for possível, os nodos de uma árvore até suas folhas. O cálculo do Fibonacci é realizado de forma individual para cada nodo até o nodo raiz, tendo como resultado o somatório do par de filhos, dessa forma o algoritmo apresenta um alto grau de paralelismo. Ao

ser expandida essa estrutura ocorre a divisão da carga de trabalho atribuída a cada ramo do grafo. A divisão da carga computacional estabelece que além do desbalanceamento existente na estrutura do grafo, onde os ramos localizados mais à esquerda são expandidos mais vezes durante a execução, seja introduzida uma carga extra de trabalho à esses nodos. Esse desbalanceamento fornece uma excelente visão da quantidade de trabalho exigido para a execução de cada ramo do grafo, estabelecendo assim um melhor mapeamento das tarefas aos PVs.

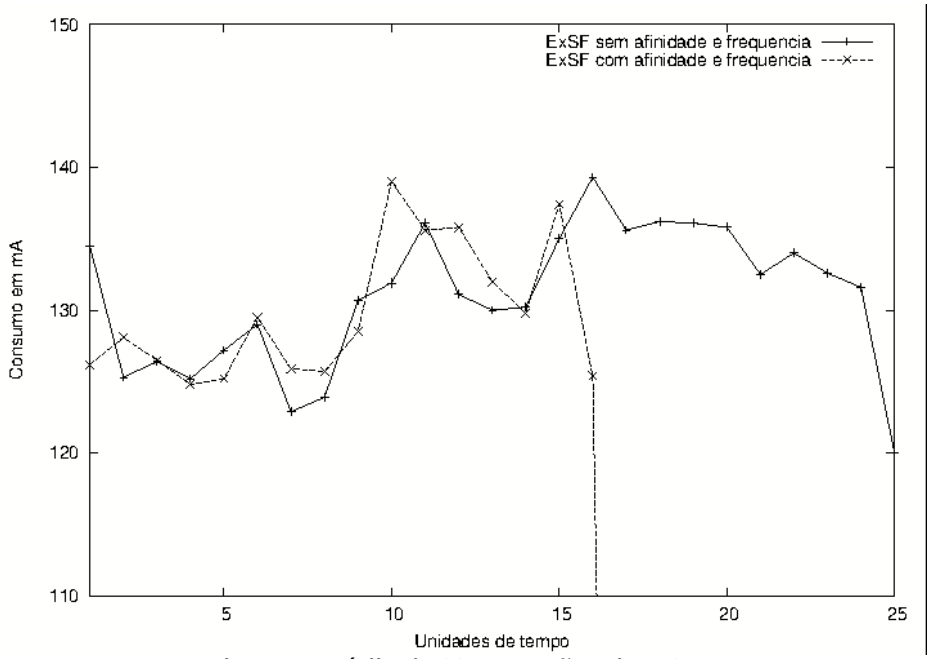

Figura 1: Média de 20 execuções de ExSF.

Figura 2: Média de 29 execuções de ExMS.

odemos observar na Figura 1 um bom índice de desempenho para esse teste a partir dos tempos de término dos dois algoritmos, esse desempenho tem impacto significativo no consumo de energia do programa, proporcionando uma economia de energia de 34.15%, dado esse obtido usando-se métodos de interpolação e cálculo da integral. O segundo algoritmo ordena um conjunto de entrada com *n * 104* números de tamanho igual a 12 dígitos cada. A implementação deste algoritmo considera que a entrada é dividida de forma balanceada entre os PVs da arquitetura. A carga computacional exercida sobre cada PV é distribuída de forma desbalanceada por dois motivos. Primeiro, há uma computação extra que cada *thread* realiza em ponto flutuante com o número cedido para a ordenação. Segundo, é considerado o número de *threads* que podem criados em cada PV controlada na etapa de divisão dos dígitos para a execução do MergeSort. A Figura 2 mostra, assim como na Figura 1, o alto desempenho da execução do MergeSort. Como visto anteriormente, no primeiro conjunto de teste, o alto desempenho das execuções é um fator importante para a redução do consumo energético, essa redução pode ser observada também na figura onde a eficiência energética para a execução do ExMS alcança um índice de 42.15% menor em relação a execução sem estratégias de controle de afinidade e frequência de operação.

5 CONCLUSÃO

Uma análise sobre os resultados revela que não há praticamente diferença na execução entre o conjunto de testes com controle de afinidade e frequência de operação e a execução sem o uso dessas estratégias de gerenciamento. Essa semelhança é observada enquanto os programas estão

executando e em parte causada pela baixa taxa de obtenção dos dados de consumo de energia gerado pelo *hardware* usado para realizar as medições durante as execuções, gerando assim grandes oscilações entres as curvas dos gráficos de consumo. Entretanto, os bons índices na economia de energia, considerando o tempo total de execução do programa, são obtidos a partir do uso eficiente dos recursos da arquitetura, esse melhor uso é proporcionado pelo gerenciamento eficaz da frequência de operação dos núcleos de processamento e pela melhor divisão das cargas computacionais dos *threads* por meio do gerenciamento de afinidade entre os processadores virtuais que compõem a arquitetura.

5 REFERÊNCIAS

HILL, M. D., MARTY, M. R. Amdahl's Law in the Multicore Age. IEEE Computer Society. Medison, Vol. 41. Pag. 33-38. 2008.

WOLF, W. The Future of Multipocessor System-on-Chip. *XLI* Design Automation Conference on DAC. San Diego, 2004.

UHRIG, S., UNGERER, T. Energy Management for Embedded Multithreaded Processors with Intergrated EDF Scheduling. ARCS. Austria, 2005.

WECHSLER, O. Setting New Standards for Energy-Efficient Performance. Inside Intel Core Microarchitecture. White Paper. Pag. 4-5. 2006.

CAVALHEIRO, G. G. H., CASPARY, L. P., CARDOZO, M. A., CORDEIRO, O. C. *VII* High Performance Computing for Computational Science. Spring-Verlag, LNCS 4395. Berlin, 2007.