XVIII CIC XI ENPOS I MOSTRA CIENTÍFICA
Evoluir sem extinguir: por uma ciência do devir

APLICAÇÃO DE AFINIDADE NO ESCALONAMENTO DE THREADS EM ARQUITETURAS MULTIPROCESSADAS VISANDO À REDUÇÃO DO CONSUMO DE ENERGIA

ARAUJO, Alan Schlindvein; CAVALHEIRO, Gerson Geraldo H.

Deptº. de Informática – DInfo/UFPel
Campus Universitário – Caixa Postal 354 – CEP 96010-900.
alansraujo.schlindvein@anahy.org, gerson.cavalheiro@ufpel.edu.br

INTRODUÇÃO

Arquiteturas multiprocessadas são caracterizadas por possuírem múltiplas unidades de processamento compartilhando acesso a um mesmo espaço de endereçamento (De Rose, 2002). Tais arquiteturas são capazes de realizar a execução concorrente de diferentes fluxos de instruções sobre unidades de processamento independentes. O espaço de endereçamento serve como substrato para a comunicação (passagem de dados) entre os processadores por meio de instruções de *load e store* na memória. Processadores multicore encaixam-se no conceito de multiprocessadores, uma vez que possuem unidades completas de processamento que fazem uso de um espaço de endereçamento comum.

A multiprogramação leve é o modelo mais adaptado à programação neste tipo de arquitetura, pois permite o desenvolvimento de programas em termos de fluxos de execução independentes (threads) que compartilham uma área de memória global (Ungerer, 2003). Neste contexto cabe ressaltar dois aspectos: (1) embora independentes, a execução dos threads deve ser coordenada entre si, pois existem dependências no acesso aos dados compartilhados expressas explicitamente no código provido pelo programador; (2) por maior que seja o número de processadores em uma arquitetura paralela, a tendência é que a concorrência descrita por um programa sempre suplante a capacidade de execução paralela disponível. Assim sendo, um mecanismo capaz de alocar fatias de tempo dos processadores aos threads se faz necessário. Este mecanismo é denominado escalonamento, havendo diferentes heurísticas para sua implementação.

O presente trabalho trata de questões relacionadas ao escalonamento de threads em arquiteturas multicore. Em particular são explorados recursos de programação deste tipo de arquitetura capazes de permitir a implementação de estratégias de escalonamento que permitam reduzir o consumo de energia e a dissipação de calor dos processadores durante a execução de um programa, sem perda significativa de desempenho. Estas implementações estão sendo realizadas pela bolsa de pesquisa no contexto do projeto Anahy (Cavalheiro, 2006). O trabalho de pesquisa referente à bolsa visa adequar velocidade de operação de cada core

em um processador multicore, considerando a carga de trabalho gerada por um programa desenvolvido em Athreads.

2. ESCALONAMENTO DE THREADS / PROCESSOS

O escalonamento é a estratégia adotada para realizar a alocação dos fluxos de execução dos programas ativos em um computador aos processadores disponíveis em uma determinada arquitetura. Existem diversas estratégias de escalonamento e todas visam maximizar a produtividade de um dado sistema computacional segundo algum critério, por exemplo, reduzir o tempo total de processamento do conjunto de programas em execução ou melhorar o desempenho da execução de um programa específico. O escalonamento é sistematicamente aplicado em nível de sistema operacional, onde a questão básica tratada é a utilização eficiente dos recursos de processamento disponíveis. Neste nível, não há distinção entre fluxos de execução e nem entre processadores, todos os elementos são tratados como iguais. Desta forma, neste nível, a maioria das implementações de escalonamento considera apenas a disponibilidade de processador para alocação de um fluxo de execução.

Quando realizado em nível aplicativo, ou seja, na camada de software do usuário, é possível considerar características próprias do programa em execução para otimizar algum índice de desempenho específico. Estas informações permitem a criação de estratégias mais elaboradas, voltadas às necessidades inerentes a uma determinada aplicação. No trabalho em desenvolvimento, os atributos considerados são aqueles que permitem identificar para um determinado thread qual o processador mais apto para suportar sua execução. A utilização prática deste atributo visa obter redução do consumo de energia e da dissipação de calor por parte dos processadores quando da execução dos threads do programa. Esta necessidade é real, pois, processadores cada vez mais robustos e com mais unidades de processamento surgem para suprir a crescente demanda por desempenho de processamento, mas com isso são destacadas questões complexas referentes ao gerenciamento de energia e a dissipação de calor por processador.

Exemplificamos o problema apresentando um estudo sobre uma aplicação sintética: o problema de Josephus (Cormen, 2001). Este problema consiste em, inicialmente, criar uma lista circular de threads e, então, realizar tantos sorteios aleatórios quanto for o número de threads nesta lista. A cada thread sorteada é atribuída uma carga de trabalho também gerada de forma aleatória.

O desempenho de uma instância desta aplicação, cujos detalhes de execução não são relevantes no presente texto, é apresentado na Figura 1.a. O gráfico desta figura apresenta a evolução no tempo de uso de dois núcleos de processamento de um computador dual-core. Por consequência, este gráfico também reflete o consumo de energia e a dissipação de calor por core. Como pode ser observado, sem o uso de afinidade, na maior parte do tempo de execução da aplicação os dois processadores permanecem em atividade com 100% do seu potencial, obviamente, as frequências das duas unidades de processamento é elevada a 100%, não trazendo resultados satisfatórios em termos de consumo de energia.

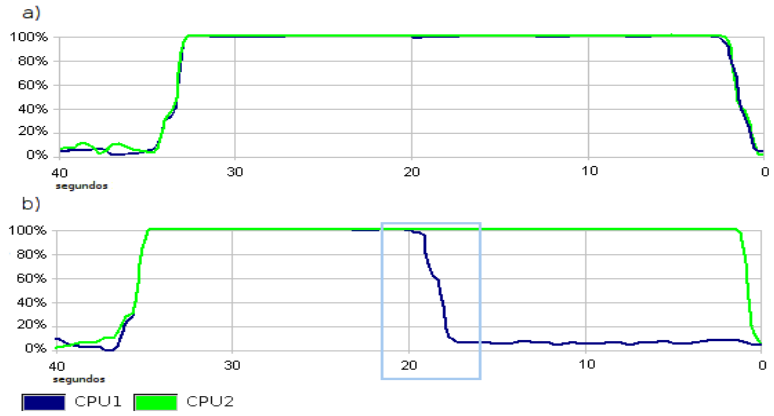

Figura 1. Porcentagem do uso dos processadores/core.

3. AFINIDADE AOS PROCESSADORES E O CONSUMO DE ENERGIA A ELA RELACIONADO

A afinidade a processador é um conceito aplicado como recurso de programação em arquiteturas de processadores modernos. Ela permite associar threads a um, ou a um grupo, de processadores. Utilizar este recurso permite explorar a localidade de referência a dados na memória cache e, com isso, podemos atribuir um thread a um processador que tenha referência aos dados disponíveis ao thread na sua cache. No entanto, o uso de afinidade não é trivial, pois exige do programador um alto grau de atenção, tanto para codificar a aplicação de seu programa em termos de concorrência, quanto para realizar a implementação desse de forma eficiente, considerando a arquitetura paralela disponível (Cavalheiro, Santos 2007). No estágio atual deste projeto, a exploração da afinidade está restrita a observação da carga de trabalho alocada a cada núcleo de processamento.

Retomando o problema de Josephus. Nesta aplicação existe um percentual de 65% de threads que possuem, individualmente, uma carga menor ou igual a 40% da maior carga alocada a um thread. Esta característica possibilita classificar os threads como “pesados” e “leves”. A Figura 1.b apresenta a execução do programa sobre a mesma arquitetura dual-core anterior empregando uma estratégia de escalonamento explorando afinidade e gerenciamento de frequência dos processadores. Neste caso, o processador 2 é dedicado à execução dos threads pesados e o processador 1 à execução dos threads leves. Enquanto o processador 2 dedicado ao processamento dos threads pesados tem sua frequência elevada, o processador 1 tem sua frequência reduzida para 63% da original.

Observando-se os gráficos das figuras 1.a e 1.b verifica-se que o segundo é produto de uma execução 6% mais lenta que a primeira. Pode-se iniciar uma discussão sobre benefícios e perdas do uso de afinidade em uma etapa posterior do trabalho. No momento destaca-se que esta situação é dependente de aplicação. Em outras palavras, o tempo de execução de um programa pode variar conforme sua implementação e como este utiliza os recursos de processamento disponíveis. Um caso de melhora de desempenho é discutido na sequência.

O clássico modelo produtor/consumidor representa uma aplicação onde um grupo de threads, denominados “produtores”, produz itens que devem ser consumidos por outro grupo de threads, denominados “consumidores”. Na implementação realizada, produtores e consumidores compartilham um buffer, no

qual é inserido e/ou consumido um item por vez, sendo que o custo de produção de um item é superior ao custo de seu consumo. O escalonamento aplicativo implementado para esta aplicação determina que todos os produtores sejam executados em um dos núcleos de processamento disponíveis (processador 2), enquanto o outro núcleo (processador 1) executa os threads consumidores em uma frequência adequada ao consumo dos itens disponíveis, evitando crescimento demasiado do buffer ou uma situação onde consumidores permanecem grande parte do tempo ociosos. Neste escalonamento, a frequência de processamento de cada núcleo pode ser ajustada conforme a necessidade de execução de cada subconjunto de threads. O escalonador determina que o núcleo dedicado a execução de produtores opere na sua frequência normal, enquanto que o processador dedicado a execução de consumidores opere em uma frequência reduzida. Como mostra a Tabela 1, os tempos médios de execução da implementação com afinidade e gerenciamento de frequência foram ligeiramente superiores aqueles em que tais recursos não foram utilizados.

Tabela 1. Média dos tempos de execução para o algoritmo Produtor/Consumidor.

10 Threads	**12 Threads**	**14 Threads**	**16 Threads**	**18 Threads**	**20 Threads**	
34.11201s	41.10931s	48.21289s	54.45442s	61.26327s	68.34528s	Com afinidade
34.21101s	41.19551s	48.63384s	54.66743s	61.51445s	68.75056s	Sem afinidade
+ 0.290%	+ 0.209%	+ 0.873%	+ 0.391%	+ 0.410%	+ 0.592%	**Desempenho**

4. CONCLUSÕES

A exploração das técnicas apresentadas neste trabalho ressalta a dificuldade de se obter desempenho e economia de energia de uma aplicação. Devemos considerar que o uso destas técnicas varia conforme a necessidade de cada aplicação e como essa atua sob os recursos disponíveis pela arquitetura. Com isso, devemos priorizar os objetivos pretendidos, adequando cada técnica no desenvolvimento do programa para obter resultados satisfatórios na execução de aplicações em termos do objetivo esperado para o algoritmo, podendo ser a economia de energia, eficiência no desempenho da execução e dissipação de calor.

No trabalho em curso não se pretende identificar os custos associados à execução de cada thread ou grupo de threads. Os esforços são voltados a obter um mecanismo de escalonamento que inclua heurísticas de distribuição de carga e determinação de frequências de processadores considerando um determinado modelo de execução.

5. REFERÊNCIAS BIBLIOGRÁFICAS

CAVALHEIRO, G. G. H. (2001). “Introdução à programação paralela e distribuída”. In. ERAD 2001. Porto Alegre – RS: SBC.

CAVALHEIRO, G. G. H.; SANTOS, R. R. (2007). “Multiprogramação leve em arquiteturas multi-core”. In: Tomasz Kowaltowski; Karin Koogan Breitman. (Org.). Atualizações em Informática 2007. Rio de Janeiro: PUC-Rio, p. 327-379.

CAVALHEIRO, G. G. H., Gaspary, L. P., Cardozo, M. A., Cordeiro, O. C. (2007) Anahy: A Programming Environment for Cluster Computing. In. High Performance

Computing for Computational Science – VECPAR 2006. Berlin: Springer.
DE ROSE, C. A. F. “Fundamentos de Processamento de Alto Desempenho”. In. ERAD 2002. São Leopoldo – RS: SBC. 2002
UNGERER, T., Robic, B., and Silc, J. (2003). “A survey of processors with explicit multithreading”. ACM Comput. Surv. V.35(1):29-63.
CORMEN, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. “Introduction to Algorithms, Second Edition. MTT Press and McGraw-Hill, 2001. Augmenting Data Structutes.