Concepção de um Núcleo Executivo para o Ambiente Dinâmico de Execução Anahy3

ARAUJO, Alan Schlindvein[1]; CAVALHEIRO, Gerson G. Homrich; DU BOIS, André Rauber

Universidade Federal de Pelotas; CDTec – Centro de Desenvolvimento Tecnológico

1 INTRODUÇÃO

Arquiteturas *multicore* são dotadas de múltiplas unidades de processamento e seu uso é extendido à aplicações de alto custo computacional. Contudo, a programação e execução neste tipo de configuração requer do programador um alto grau de conhechimento sobre multiprogramação.
Com isso, é necessário construir ferramentas que visam simplificar a programação neste tipo de arquitetura. Em geral, tais ferramentas fornecem meios para que o programador possa descrever seu programa por meio de fluxos de execução chamados *threads*. Estes *threads* são concorrentes e compartilham um mesmo espaço de endereçamento. Este espaço de endereçamento serve como substrato de comunicação para que *threads* transfiram dados entre si.

Conforme o programa evolui, estes *threads* são criados, sincronizados e destruídos a partir da especificação feita pelo programador, essas operações são realizadas sob um grafo dirigido acíclico (DAGs). O desempenho dos ambientes *multithread* dinâmicos depende do quão eficiente é o escalonamento realizado sobre a estrutura de controle das dependênias entre os *threads*. A partir disso, o ambiente de programação deve ser suficientemente robusto para que operações sobre a estrutura de controle não agreguem impacto negativo no desempenho da aplicação.

A ferramenta proposta neste trabalho é sensível ao consumo de energia e, portanto, questões relacionadas à práticas de computação verde (SCHAEFFER, 2012) são consideradas. Processadores *multicore* modernos disponibilizam, em sua arquitetura, diversas facilidades para controlar o modo de execução de cada *cores* (CHEN, 2007). O gerenciamento de afinidade permite especificar um *cores* responsáveis pela execução de um ou mais processos. O controle da frequência, em sistemas dependentes de eficiência energética, possibilita adaptar a velocidade com que as funções em cada *core* são executadas.

O uso combinado destes recursos permite que possam ser tomadas decisões de escalonamento que reduzam o consumo de energia necessário para executar o programa. No entanto, para que tais recursos sejam utilizados de forma eficiente, é necessário que exista conhecimento sobre o comportamento do programa em execução, para que decisões de escalonamento sejam feitas considerando a evolução do programa. Assim sendo, o ambiente de programação deve oferecer uma interface simplificada ao programador, para que este seja capaz de explorar o uso de afinidade e controle de frequência de forma eficiente. Em outras palavras, o ambiente dever oferecer abstrações na interface que sejam capazes de ajudar o programador a fornecer informações específicas sobre seu programa, como custo computacional ou custo de comunicação entre tarefas, e, então, utilizar heurísticas que considerem estas informações para escalonar e execução tarefas o mais eficientemente possível.

[1]Bolsista BIC FAPERGS

Neste sentido, a construção de um núcleo de execução para o ambiente de programação chamado Anahy3. O núcelo da ferramenta fornece meios para que o programador possa facilmente lançar suas atividades concorrentes sem se preocupar com a arquitetura em si. O ambiente oferece também suporte à coleta de dados específicos do programa e mecanismos de escalonamento com o objetivo de reduzir o consumo de energia. Isso, devido a um gerenciameno eficiente dos recursos disponíveis em *hardware*, sem agregar impacto negativo no desempenho da aplicação.

2 Construindo uma Estratégia Básica para Economia de Energia

O uso dos recursos de afinidade e de controle de frequência visam a construção de uma camada em *software* capaz de prover índices de eficiência energética satisfatórios, desde que as heurísticas utilizadas pelo ambiente sejam empregadas de maneira correta, sem introduzir *overhead* na execução do programa.

2.1 Escalonamento Dinâmico de Voltagem

Toda a transição em um circuito digital gera consumo de energia, pois capacitância existente nos circuitos digitais que consome potência ao carregar e descarregar o *hardware*. DVS (*Dynamic Voltage Scaling*) (BURD, 2000) é uma técnica bem consolidada no projeto de arquiteturas sensível ao consumo de energia. Esta técnica consistem em explorar a relação entre voltagem e frequência de operação, conservando energia quando necessário, ou seja, dinamicamente é adaptada a voltagem de um circuito em função da frequência de operação dos *cores*.

No contexto deste trabalho, esse recurso serve para que sejam atribuídas diferentes frequências de operação aos *cores*, a partir da carga de trabalho exigida a cada um, ou seja, *cores* que possuam muitos *threads* operam com velocidades maiores do que cores que não não possam tanto trabalho.

2.2 Gerenciamento de Afinidade

A técnica de controle de afinidade utiliza um agente de decisão para realizar o mapeamento de *threads* sobre os processadores. No início do escalonamento toda tarefa, sendo ela crítica ou não, é alocada de acordo com a decisão do escalonador de afinidade. Contudo, tarefas não críticas ao sistema realizam muitas trocas de contexto, neste caso, não se deve considerar apenas a afinidade destas tarefas como o fator determinante para seu mapeamento.

O agente responsável pelas tomadas de decisão de afinidade, precisa monitar a carga de trabalho do processador, antes de atribuir uma tarefa a ele, evitando com isso, a atribuição de tarefas a processadores altamente carregados. O monitoramento da carga computacional, em cada processador, é realizado a partir da transferência periódica de tarefas no sistema. O uso de afinidade pelo núcleo de escalonamento permite explorar a localidade de dados nos *caches* dos *cores*, isso é observado quando um *thread* que parou sua execução por algum motivo, quer retomar a execução no processador que o estava executando anteriormente, elemininando desta forma a necessidade de realocar seu contexto de continuação em outro processador.

3 Arquitetura do Ambiente

O ambiente Anahy3 implementa suas próprias unidades de execução chamadas VPs. Cadas VP possui uma lista local de *threads* internamente chamados de *jobs.* Isso permite que um VP trabalhe de maneira independente durante sua execução, criando e sincronizando seus próprios *jobs*. Contudo, o gerenciamento dos VPs é tarefa do componente chamado DaemonAnahy. O DaemonAnahy é o coração do sistema, ele cria e destrói VPs, atribui frequências de operação aos *cores* e realiza o escalonamento no segundo nível, onde ocorre o mapeamento dos *threads* sistema aos núcleos de execução. Este *daemon* também fornece um meio para que VPs ociosos possam requisitar trabalho. O roubo de trabalho é centralizado. Isso facilita a implementação da estratégia de roubo, bem como as estruturas de dados não bloqueantes utilizadas pelos VPs para manterem seus *threads* locais.

A Figura 1 apresenta uma visão geral da ferramenta de programação, bem como a estrutura do núcleo de execução. O ambiente fornece quatro primitivas para básicas para a descrição do programa *multithread*. `init` é utilizada para configurar o ambiente com dados específico do programa, fornecidos pelo programador, como número de processadores vituais (VPs), custo computacional das tarefas e do programa e número de *joins* de cada *threads.* `create` e `join` são as primitivas principais do ambiente utilizadas para criar e sincronizar, respectivamente, um *thread* no sistema. `terminate` finaliza a execução do programa, desalocando espaço em memória e restaurando a política de escalonamento de frequência do sistema operacional.

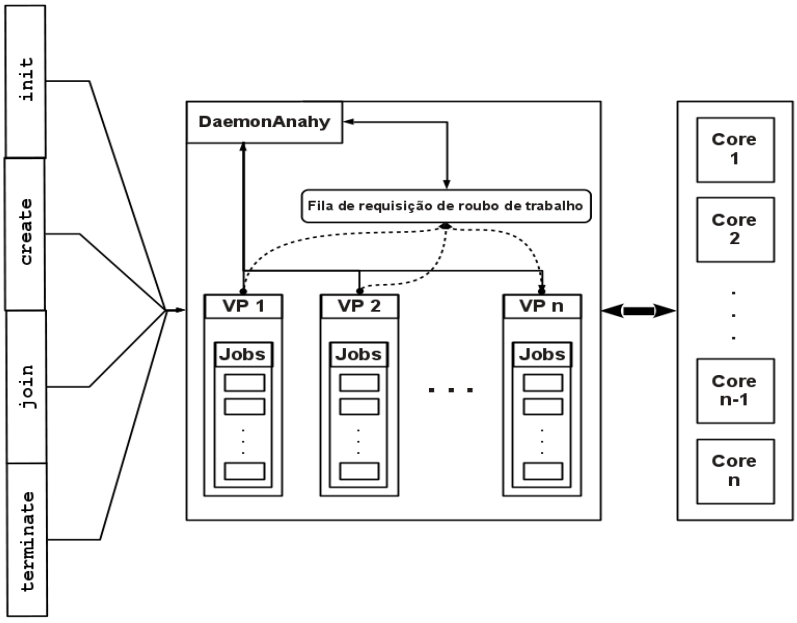

Figura 1: Arquitetura do Anahy3.

4 Roubo de Trabalho

A estratégia de escalonamento dá-se da seguinte maneira: quando VP realiza uma busca mal sucedida em sua lista local, ele envia uma requisição de roubo de trabalho ao *daemon* colocando-se em uma lista de espera. Caso o *daemon* encontre trabalho, o VP é retirado da fila de espera e, então, executa a tarefa roubada. Caso o *daemon* não

encontre trabalho para ser roubado o VP permanece na fila até que um novo trabalho seja criado. O trabalho principal do *daemon* é monitorar esta fila, "acordando" VPs que estejam aguardando trabalho. O término do programa, dá-se quando a primitiva `terminate` é chamada e todos os VPs estão aguardando. O de um *daemon*, como descrito na Seção 3, centraliza o roubo de trabalho. Desta forma a estratégia do roubo parte do princípio de que, durante toda a execução do programa, não mais do que dois *threads* fazem operações de leitura e escrita na lista local de cada VP, diminuindo assim o risco de corromper a estrutura de controle dos *threads* a partir do número de operações sobre ela.

5 CONCLUSÃO

O ambiente proposto neste trabalho é de extrema importância para o entendimento de como abstrações em *software*, mais precisamente ambientes de programação e execução dinâmicos, podem facilitar a construção de algoritmos paralelos eficientes, sem exigir grandes conhecimentos sobre multiprogramação do programador. Este trabalho encontra-se em fase de construção, não contemplando ainda resultados finais. Porém, diversas partes do ambiente já passaram por etapas de testes individuais, obtendo resultados satisfatórios em termos de desempenho. Estes testes são previstos nas etapas do projeto e durante sua construção com o objetivo de eliminar quaisquer possíveis gargalos de desempenho na versão final do ambiente. A expectativa é, que a junção das diferentes partes do ambiente tenha o comportamento esperado quanto à eficiência do consumo de energia e de desempenho. Os conceitos apresentados neste trabalho são bem consolidados na área, partes dos experimentos realizados à respeito da eficiência energética podem ser encontrados em (ARAUJO, 2012).

6 REFERÊNCIAS

SCHAEFFER, Donna; RAGHAVAN, Srinivasan; CAMERLINCK, Tom; MCKENZIE, Water. Going Green with Computing. **Journal of Computing Sciences in Colleges**, v. 27, n. 3, p. 30, 2012.

CHEN, Jian Jia; YANG, Chuan Yue; KUO, Tei Wei; SHIH, Chi Sheng. Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems. **Asia South Pacific Design Automation Conference ASP-DAC '07**, IEEE Computer Society, p. 342-349, 2007.

BURD, Thomas D.; BRODERSEN, Robert W. Design Issues for Dynamic Voltage Scaling. **International Symposium on Low Power Electronics and Design ISLPED '00**, ACM Special Interest Group on Design Automation, p. 9-14, 2012.

ARAUJO, Alan S.; CAMARGO, Cícero A. S.; NACHTIGALL, Matheus G.; FAVARETTO, Rodolfo M.; DU BOIS, André R.; CAVALHEIRO, Gerson G. H. Design Issues for Dynamic Voltage Scaling. **12ª Escola Regional de Alto Desempenho ERAD/RS**, p. 145-148, 2012.