`Athreads` - Implementação do modelo de programação Anahy Vanilla

Elvio Viçosa Junior, Douglas E. Rosa, Lucas D. Fonseca,
Cícero Augusto S. Camargo, Gerson Geraldo H. Cavalheiro

Departamento de Informática
Universidade Federal de Pelotas (UFPel)
Pelotas – RS – Brasil

{elvio.vicosa, douglas.erosa, lucas.fonseca}@anahy.org

{cicero.camargo, gersonc}@anahy.org

Resumo. *Este trabalho apresenta a implementação do protótipo* `Athreads`*, detalhando sua API e modelo de execução.* `Athreads` *é direcionado para explorar a concorrência de tarefas em arquiteturas paralelas (SMPs e multi-core) com memória NUMA. Este ambiente permite a descrição da concorrência da aplicação com independência de arquitetura, possibilitando a portabilidade de desempenho da aplicação entre arquiteturas paralelas com diferentes capacidades de processamento, sem a necessidade de modificação de código. O ambiente teve seu desempenho avaliado através de 3 variações de uma aplicação denominada miner, com o objetivo de analisar o efeito sobre os tempos de execução.*

1. Introdução

Com a popularização de processadores multi-core houve um aumento de ofertas de multiprocessadores (tanto arquiteturas UMA quanto NUMA) no mercado. Uma das questões a serem tratadas no desenvolvimento de tais arquiteturas é como obter uma implementação que seja ao mesmo tempo eficiente e escalável com o número de processadores disponíveis. Uma das possíveis solução para este problema é utilizar mecanismos de escalonamento que explorem eficientemente os recursos do hardware disponível [Kwok99]. Esta estratégia é explorada em diversos modelos de ambientes para processamento de alto desempenho, como Anahy [Cor05].

Este artigo apresenta um caso de estudo com o protótipo `Athreads`, uma implementação de Anahy Vanilla[1], que oferece uma interface de programação baseada no padrão POSIX threads (Pthreads). Esta ferramenta oferece uma abstração de programação multithread que permite obter informações sobre dependências de dados entre tarefas para explorar a localidade de referência a dados por parte dos threads em execução.

Na próxima seção a biblioteca de programação multithread Athreads é introduzida. A Seção 3 apresenta um estudo de caso e a avaliação de desempenho de Athreads em uma arquitetura NUMA. A Seção 4 conclui o trabalho.

2. Athreads: Anahy threads

Anahy é um modelo de programação e execução concebido para suportar o desenvolvimento de interfaces de programação para computadores paralelos. Anahy Vanilla é o subconjunto de Anahy oferecendo uma interface de programação baseada em

[1] www.anahy.org

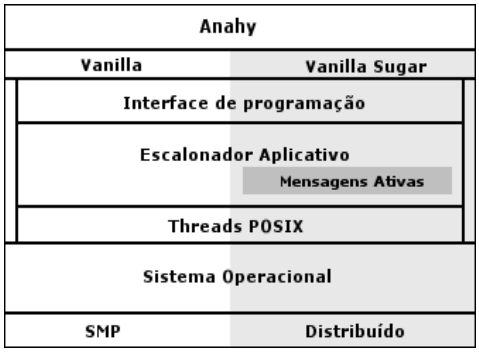

Figura 1. Estrutura em camadas de Anahy Vanilla e Anahy Vanilla Sugar

multiprogramação leve (multithreading) e suporte de execução sobre arquiteturas multiprocessadas (UMA ou NUMA). Athreads (Anahy threads) é uma implementação de Anahy Vanilla segundo o padrão Pthreads.

A concorrência em Anahy Vanilla é descrita em termos de criação e sincronização de threads por invocações de operações *split/join*, ambas invocadas no contexto de execução de um thread. A invocação à operação *split* implica na criação de um novo thread, já uma invocação à operação *join* implica na sincronização do término de um thread com a execução da próxima instrução no corpo do thread corrente. O acompanhamento, pelo núcleo de execução de Anahy Vanilla, da invocação destes serviços permite identificar não apenas a relação de precedência entre os threads criados, mas também o grafo de fluxo de dados entre tarefas. Cabe destacar o conceito de thread apresentado em Anahy Vanilla: um thread consiste em uma seqüência de tarefas que devem ser executadas em uma ordem pré-definida. O conceito de tarefa é herdado de Anahy: uma tarefa recebe um conjunto de dados de entrada e produz, ao seu final da execução de uma seqüência de instruções, um conjunto de dados de saída.

Além de manter um grafo descrevendo a relação de precedência entre threads, o núcleo de execução provê *processadores virtuais* (PVs) para suportar a execução dos threads definidos pelo programa em execução e implementa um mecanismo de escalonamento responsável por alocar threads aos PVs. Este escalonamento respeita ordem de precedência especificada no grafo e garante que, uma vez que um thread é lançado sobre um PV, o thread irá executar sobre este mesmo PV até ser concluído. Como conseqüência, o número de threads em execução concorrente é limitado pelo número de PVs criados pelo núcleo de execução.

A implementação de Anahy Vanilla em `Athreads` segue o padrão Pthreads, redefinindo funcionalidades para um conjunto de serviços desta interface de programação. As operações *split/join* são implementadas pelas primitivas:

```
int athread_create(athread_t *thid, athread_attr_t *atrib,
                   void *(*func)(void *), void *entrada);
int athread_join(athread_t thid, void **saida);
void *func(void *entrada) { /* ... */ return saida;}
```

A invocação à primitiva `athread_create` implica na criação de um novo thread. Este thread é adicionado ao grafo de precedência, sendo considerado apto a ser

executado pois os parâmetros necessários a sua execução, descritos em `entrada`, já foram produzidos. A seqüência de instruções que é utilizada na execução do novo thread é definida no corpo da função `func`. O parâmetro `atrib` oferece para o programador a possibilidade de definir atributos de execução do thread criado, como o número máximo de operações de *join* que o thread pode suportar. Após a execução da primitiva `athread_create`, a memória endereçada por `thid` é atualizada com o descritor do novo thread criado.

Para obter o resultado de um thread, utiliza-se a instrução `athread_join`, especificando através `thid` qual thread deve ser sincronizado. Se uma chamada à `athread_join` for feita em um thread que ainda não terminou sua execução, o ponto de chamada desta execução ficará bloqueado até que o thread solicitado termine. Após o término do thread, o resultado é disponibilizado através de uma área global da memória, sendo o endereço desta área atualizado no parâmetro `saida`. É importante notar que a estratégia de escalonamento não permite que o PV fique ocioso enquanto o thread que executou a operação join permanece bloqueado: é implementada uma heurística que reutiliza o PV para otimizar a execução.

3. Estudo de caso: *miner*

O *miner* é uma aplicação recursiva que gera um grafo de tarefas desbalanceado. Cada mineiro recebe como parâmetro de entrada uma carga de trabalho `C` e deve produzir o resultado da mineração. Cada mineiro cria, para auxilia-lo, outros m mineiros, com cargas de trabalhos `C-1`, `C-2`, ..., `C-`m. No presente estudo de caso, cada mineiro cria dois ajudantes ($m = 2$), como ilustra a Figura 2. Note que $m \leq C - 1$ para que o mineiro corrente possa executar parte do trabalho de mineração.

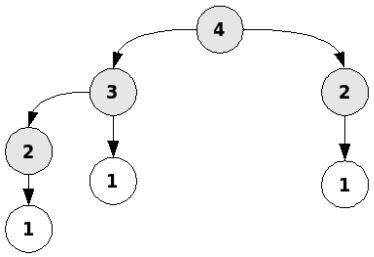

Figura 2. Grafo do aplicativo *miner* para n=4

Para medir o desempenho e o comportamento desta aplicação, foram utilizadas três variações da aplicação *miner*. As versões foram classificadas como: *leftmost*, *depthmost*, e *rightmost*. Na versão *leftmost*, cada *miner* que precisa de ajuda para computação ($n > 1$), cria dois novos *miners*: $n-1$ é calculado concorrentemente, enquanto $n-2$ é calculado seqüêncialmente. A versão *rightmost* trabalha de forma inversa. O valor de `n-1` é calculado seqüêncialmente, enquanto o valor de $n - 2$ é calculado concorrentemente. A versão *depthmost* calcula tanto $n - 1$ e $n - 2$ de forma concorrente.

Os testes de desempenho da aplicação foram executado sobre uma arquitetura com 16 processadores e acesso à memória não uniforme(NUMA). Abaixo são apresentados os gráficos de tempos de execução e *speedup*.

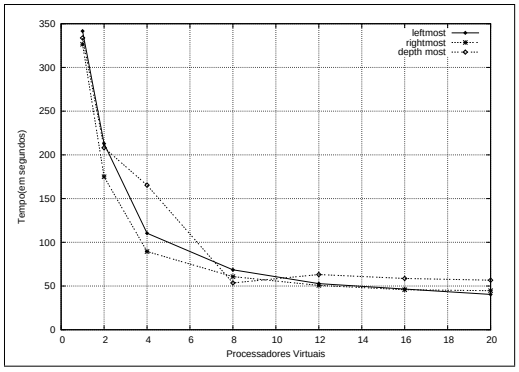

Figura 3. Tempo de execução

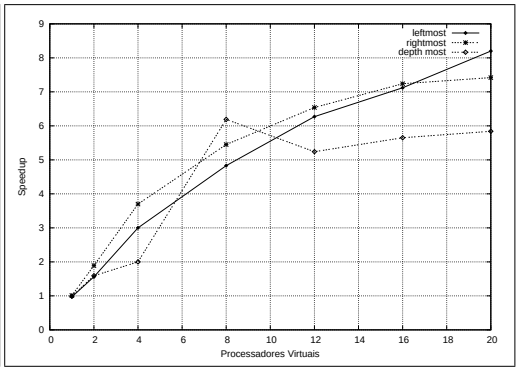

Figura 4. *Speedup*

4. Conclusão

A criação de um ambiente que permite reduzir o custo de desenvolvimento e manter a portabilidade do desempenho entre as diferentes arquiteturas, é o principal objetivo do ambiente Anahy Vanilla.

Os resultados obtidos através do caso de estudo, mostraram que o comportamento do ambiente e os tempos obtidos foram satisfatórios. Este caso de estudo foi testado em arquiteturas de diferentes configurações, variando em 2, 4, 8 e 16 processadores. Em todas as execuções foram obtidos alto-desempenho, utilizando todos os recursos disponíveis e sem a necessidade de alteração do código. Através do gráfico mostrado na Figura 3, pode-se observar que a versão *leftmost* obteve o melhor desempenho médio na execução. Este resultado pode ser explicado levando em conta o comportamento do aplicativo *miner*. Como descrito anteriormente, o grafo que descreve a recursividade deste aplicativo é desbalanceado, possuindo sempre o lado esquerdo maior que o lado direito, exigindo desta maneira, uma maior carga computacional para ser completamente avaliado.

Para os próximos trabalhos, o ambiente Anahy Vanilla irá incorporar mecanismos de sincronização padrão em threads POSIX, verificação do impacto de diferentes estratégias de escalonamento sobre o grafo que descreve as dependências entre as tarefas. Também será verificado o efeito da utilização de técnicas de afinidade das threads responsáveis pelos PVs com processadores(ou *cores*), com o objetivo de otimizar os acessos à *cache*. Também serão otimizados os mecanismos no ambiente Anahy Vanilla Sugar, voltado para programação concorrente distribuída.

Referências

Cordeiro, Otávio et al(2005). Exploiting Multithreaded Programming on Cluster Architectures *HPCS*.

Kwok, Y. K. e AHMAD. I. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406-471 In VII High Performance Computing for Computational *Science*, Berlin 2006 Springer-Verlag(LNCS 4395) 1999