Programação Paralela com *threads* Anahy*

Epifanio Dinis Benitez†, Gerson Geraldo H. Cavalheiro

[1]Programa Interdisciplinar de Pós-Graduação em Computação Aplicada
Universidade do Vale do Rio dos Sinos
Avenida Unisinos, 950 – 93.022-000 São Leopoldo, RS

epifanio@exatas.unisinos.br, gersonc@unisinos.br

Abstract. *Anahy programming/execution environment is a tool for High Performance Computing (HPC) that exploit efficiently the resources present in parallel architectures. As a library of light process, it's structure is composed by virtual processors, dependecy graph and a scheduling kernel. This structure makes possible to implement higly parallel applications with the Application Programming Interface available. This paper describes Anahy execution model and compares it with Pthreads execution model.*

Resumo. *O ambiente de programação/execução Anahy é uma ferramenta para o PAD (processamento de alto desempenho) que explora de forma eficiente os recursos presentes em arquiteturas paralelas. Como bibilioteca de processos leves, sua estrutura está baseada em processadores virtuais, grafo de dependência de dados e um núcleo executivo de escalonamento. Isto permite implementar de forma eficiente aplicações altamente paralelas através de sua interface aplicativa. Este trabalho descreve a estrutura de Anahy e mostra um comparativo entre os modelos de execução das threads do padrão POSIX (*PThreads*) e das threads Anahy.*

*Projeto Anahy – CNPq (55.2196/02-9). Este trabalho foi parcialmente desenvolvido em colaboração com a HP Brasil P&D

†BIC/UNISINOS

1. Introdução

Universidades e centros de pesquisa que trabalham com aplicações de alto custo computacional das áreas da biologia, física e computação científica, buscam alcançar melhores resultados através de máquinas com clock mais alto. Pelo seu relativo baixo custo, as arquiteturas multiprocessadas (SMP) se tornaram uma ótima opção para obter ganho de desempenho. Isto motiva os pesquisadores a transformarem seus códigos seqüenciais em códigos paralelos, de forma a utilizar os recursos disponíveis e obter ganho de desempenho diminuindo o tempo de execução de suas aplicações. Para que isto seja possível, busca-se codificar o programa paralelo baseando-se no mapeamento da concorrência da aplicação no paralelismo do hardware. Porém, aplicações que realizam uma grande quantidade de operações com pouco volume de dados, isto é, com granularidade fina, a concorrência da aplicação supera em várias vezes o paralelismo disponível na arquitetura, limitando o ganho de desempenho esperado.

A solução a este problema é um modelo de execução de *threads* que execute a maior quantidade de fluxo de execução por "fatia" de tempo atribuída pelo escalonador. No modelo tradicional, com as *threads* do padrão POSIX [American National Standards Institute, 1994], existem duas limitações no ganho de desempenho para aplicações de granularidade fina: i) cada *thread* executa somente um fluxos de execução e ii) a troca de contexto que ocorre ao criar/terminar as *threads* gera um alto *overhead*. A primeira limitação se torna um problema quando o fluxo de execução – ou tarefa – exige um tempo de execução menor que o tempo disponibilizado pelo escalonador do sistema operacional. A segunda limitação, neste modelo de execução, é um problema inevitável e se torna grave quando esse tempo de criação/término das *threads* é maior que o tempo de execução do fluxo que elas devem executar.

Anahy, como biblioteca de processos leves [Cavalheiro et al., 2003], apresenta um modelo de execução que i) executa a maior quantidade de tarefas na mesma "fatia" de tempo atribuída pelo escalonador e ii) minimiza ao máximo o *o*verhead gerado durante a troca de contextos. Isto é possível através dos componentes que formam a estrutura de Anahy e são eles: processadores virtuais, núcleo executivo de escalonamento e um grafo de dependência de dados. A interação entre estes componentes se dá pelo gerenciamento do grafo de dependência de dados, – criado através da concorrência descrita pelo programador –, pelo núcleo executivo de escalonamento. O núcleo de escalonamento, ou *Kernel* de Anahy, possui uma política de escalonamento encarregada de realizar a atribuição de tarefas aos processadores virtuais disponíves. O programador pode alterar a política de escalonamento, no código de Anahy, e também pode variar o número de processadores virtuais disponíveis na biblioteca. Anahy oferece acesso aos serviços através de uma interface aplicativa, disponível ao programador.

Este trabalho apresenta os componentes de Anahy, como biblioteca de processos leves [1] e um comparativo com o modelo tradicional do padrão POSIX para *threads*. Na seção seguinte, Anahy é descrito em função de seus componentes. A Seção 3 descreve os modelos de execução de Anahy e *Pthreads* e na Seção 4 as conclusões.

[1] código fonte e documentação disponí vel para download em http://anahy.org

2. Anahy

Anahy é um ambiente de programação/execução, desenvolvido dentro do Projeto Anahy e distribuído sobre licença GPL, que busca otimizar a execução de aplicações com granularidade fina, de forma a obter ganho de desempenho através da utilização eficiente dos recursos disponíveis em hardware. Através de sua API, é possível descrever a concorrência da aplicação, que será mapeada a um grafo de dependências de onde o *Kernel* de Anahy irá escalonar as tarefas aos processadores virtuais. Nas sub-seções seguintes se encontra a descrição dos componentes mencionados, que diferem Anahy de *Pthreads*.

2.1. Processadores Virtuais (PVs)

As aplicações concorrentes, em sua maior parte, possuem mais atividades do que o paralelismo real da arquitetura alvo. Anahy disponibiliza ao programador os PVs, que formam uma arquitetura *virtual* multiprocessada dotada de memória compartilhada, representada na Figura 1. Através dos PVs o programador pode definir um número de atividades concorrentes maior que o paralelismo real da arquitetura. Desta forma torna-se possível a portabilidade de código e desempenho [Dall'Agnol et al., 2003].

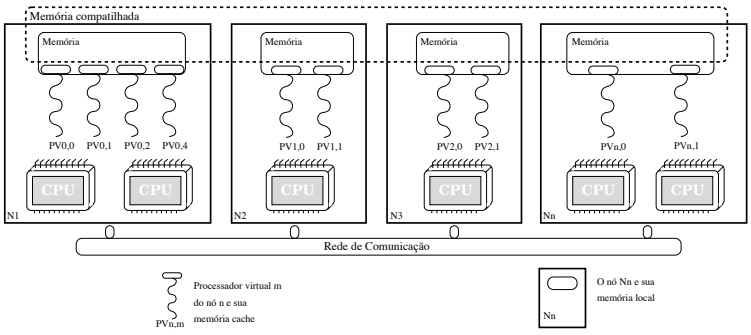

Figure 1: Modelo lógico e físico da arquitetura de suporte à Anahy

Durante a execução do ambiente Anahy, cada PV recebe do escalonador o maior número de tarefas possível, aproveitando ao máximo a "fatia" de tempo disponibilizada pelo escalonador. Os PVs, durante a execução de um fluxo de operações, não irão tratar nenhuma sinalização e quando algum ficar ocioso, ele será ativado pelo escalonador assim que houver alguma tarefa a ser executada. O programador pode definir o número de PVs a serem utilizados, aumentando os recursos da arquitetural *virtual*. Para cada processador virtual (*thread de nível aplicativo*) criado, uma *thread* no nível sistema é criada.

2.2. Núcleo executivo de escalonamento

O núcleo executivo é responsável pelo escalonamento e sua é função mapear as tarefas da aplicação à arquitetura virtual de Anahy. Possui uma política de escalonamento que realiza o balanceamento de carga, para que as tarefas sejam distribuídas igualmente entre os PVs. O programador pode introduzir novas políticas de escalonamento para alterar o balanceamento de carga, mas isto deve ser realizado em tempo de desenho (alterando o código fonte do escalonador de Anahy). Ao contrário de PThreads, onde o escalonamento de tarefas é realizado no nível sistema, em Anahy, o escalonamento é realizado a nível de aplicativo (*cf.* [Cavalheiro, 1999], cap.2).

O grafo de dependência das tarefas, – sobre o qual atua o escalonador –, pode ser visto na Figura 2. Cada vértice do grafo é uma *thread* Anahy e as arestas representam a

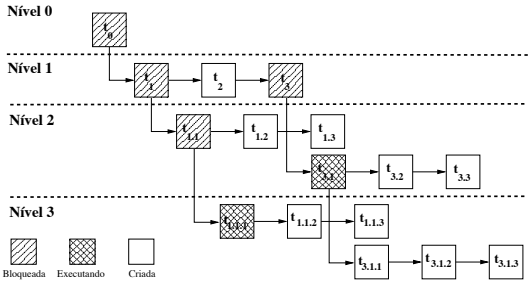

Figure 2: Grafo de criação de threads Anahy e relações de precedência de execução

dependência entre as *threads*. Para o melhor entendimento, este grafo deve ser analisado em cada nível.

No nível 0 existe somente uma tarefa, pois ela é o "pai" de todas as tarefas nos níveis seguintes e a continuidade de sua execução se dará quando todas as tarefas "filhas", vistas nos níveis 1, 2 e 3, estiverem terminado. Então quando a tarefa "pai" $\mathcal{T}$ solicitar o resultado de sua tarefa filha $\mathcal{T}$, na verdade $\mathcal{T}$ estará solicitando o último resultado recebido pela tarefa filha $\mathcal{T}$. Para facilitar a visualização da dependência de dados entre as tarefas, a ordem de execução neste grafo é realizada da direita para a esquerda e de baixo para cima. Cada tarefa presente no grafo está representada por tons de cinza para que possam ser identificadas as tarefas **criadas**, as que estão **executando** e as que estão **bloqueadas**.

2.3. Interface Aplicativa (API)

Os serviços que permitem ao programador descrever a concorrência da aplicação se encontram disponíveis através de uma API. Por questões de compatibilidade, a sintaxe utilizada por Anahy segue o padrão POSIX para *threads*. Desta forma, para transformar uma aplicação desenvolvida com PThreads para Anahy basta alterar o nome da chamada das rotinas de sincronização das *threads*. O sincronismo permitido em Anahy está formado basicamente por operações de *fork* e *join*, que correspondem as operações de criação e espera por término de *threads* respectivamente. Os operadores do padrão POSIX *pthread_create* e *pthread_join* são substituídos pelo seu equivalente em Anahy, e estão exemplificados abaixo por:

```
int athread_create( athread_t *th,
                    athread_attr_t *atrib,
                    void * (*func) (void *),
                    void *in );
int athread_join( athread_t th, void **res);
```

Para as *threads* Anahy, o programador pode utilizar alguns atributos extras, entre eles: *athread_attr_setjoinnumber* e *athread_attr_getjoinnumber*, que servem para explicitar e recuperar a quantidade de operações join podem ser realizadas sobre uma determinada tarefa (*thread* Anahy); *athread_attr_setdatalen* e *athread_attr_getdatalen*, que servem para declarar e recuperar o tamanho dos dados das tarefas, entre outras. Atualmente estão sendo testadas rotinas para o envio e recebimento de tarefas entre nodos de um aglomerado de computadores.

3. Modelos de Execução

Esta seção complementa a descrição de Anahy através da comparação seu modelo de execução com o de PThreads. Além de uma descrição, algumas figuras são utilizadas para visualizar estes modelos com o intuito de complementar o que está descrito.

3.1. Pthreads

A biblioteca de *threads* do padrão *POSIX* [American National Standards Institute, 1994], mais conhecida pela implementação da Linux Threads, apresenta um modelo de execução onde cada *thread* consiste em um novo fluxo de execução (Figura 3). A cada novo fluxo de execução criado, é gerado um contexto que deverá ser gerenciado pelo escalonador do sistema operacional. O gerenciamento consiste em guardar o contexto quando o fluxo não estiver em execução e recuperá-lo, quando o fluxo for escalonado novamente, gerando um *overhead* na troca de contextos. Um fluxo em execução, em *Pthreads*, significa que somente uma atividade será realizada dentro de uma "fatia" de tempo. E ainda, quando o fluxo realizar por completo sua atividade, isto é, quando ele terminar, o resultado por ele produzido poderá ser recuperado somente uma vez, obrigando alocar estruturas auxiliares caso o resultado necessite ser recuperado mais vezes.

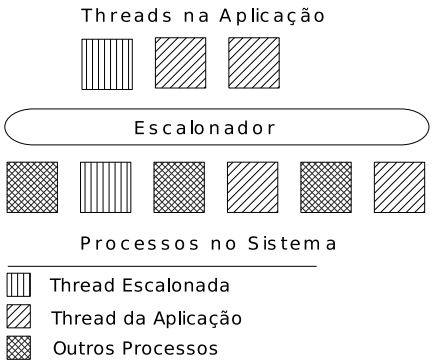

Figure 3: Modelo de execução das *threads* POSIX

Na Figura 3, vemos que para cada thread criada na aplicação, uma thread é criada no sistema, portanto a estratégia utilizada é *one-to-one* (um para um).

3.2. Anahy *threads*

Uma thread Anahy difere de uma Pthread em sua composição. Uma Pthread é composta por somente um fluxo de execução, já uma thread Anahy consiste no encapsulamento das tarefas explicitadas na aplicação (através da API). Este encapsulamento é realizado pelo ambiente Anahy seguindo a relação de paralelismo de controle existente entre elas, como pode ser visto na Figura 4.

Na Figura 4, vemos que as *threads* declaradas na camada da aplicação são mapeadas nos PVs através do núcleo executivo. Como cada PV pode receber mais de uma tarefa para ser executada, a estratégia utilizada é *one-to-many* (um para muitos). Este é o principal diferencial entre as *threads* Anahy e PThreads.

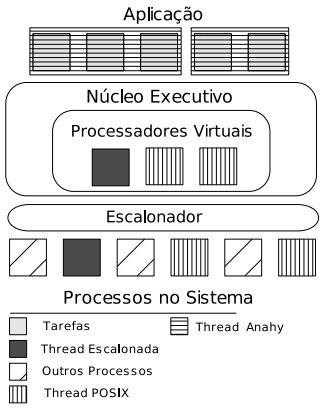

Figure 4: Modelo de execução do Anahy

4. Conclusão

Com este trabalho é possível concluir que para obter ganho de desempenho ao paralelizar uma aplicação seqüencial não basta simplesmente codificar, como também é necessário identificar a granularidade da aplicação para buscar um modelo de execução que permita obter o ganho desejável.

Anahy, como biblioteca de processos leves, oferece uma arquitetura e um conjunto de serviços que permitem utilizar de forma eficiente os recursos existentes em arquiteturas paralelas, para aplicações com granularidade fina. Sua estrutura e seu modelo de execução geraram ótimos resultados, que podem ser vistos em [Benitez et al., 2004]. Atualmente, o Projeto Anahy desenvolve uma biblioteca para programação paralela e distribuída em aglomerados de computadores. Os resultados atuais estão em processo de publicação e por isso não foram citados neste trabalho.

References

American National Standards Institute (1994). *IEEE standard for information technology: Portable Operating Sytem Interface (POSIX). Part 1, system application program interface (API) — amendment 1 — realtime extension [C language]*. IEEE Computer Society Press.

Benitez, E. D., Dall'Agnol, E. C., Real, L. C. V., Junior, M. A. C., and Cavalheiro, G. G. H. (2004). Avaliação de desempenho de anahy em aplicações paralelas. In *Anais digital do III Workshop em Performance de Sistemas Computacionais*, Salvador, Brasil.

Cavalheiro, G. G. H. (1999). *Athapascan-1: Interface générique pour l'ordonnancement dans un environnement d'exécution parallèle*. Thèse de doctorat, Institut National Polytechnique de Grenoble, Grenoble, France.

Cavalheiro, G. G. H., Real, L. C. V., and Dall'Agnol, E. C. (2003). Uma biblioteca de processos leves para a implementação de aplicações altamente paralelas. In *Anais do IV Workshop em Sistemas Computacionais de Alto Desempenho*, São Paulo, Brasil.

Dall'Agnol, E. C., Real, L. C. V., Benitez, E. D., and Cavalheiro, G. G. H. (2003). Portabilidade na programação para o processamento de alto desempenho. In *Anais do IV Workshop em Sistemas Computacionais de Alto Desempenho*, São Paulo, Brasil.