ApenMP: um suporte à OpenMP para o modelo Anahy

Cristian Fernando F. Castañeda, Gerson Geraldo H. Cavalheiro

Departamento de Informática – Universidade Federal de Pelotas
Pelotas – RS – Brasil

{ccastaneda_ifm, gerson.cavalheiro}@ufpel.edu.br

Resumo. *Este trabalho apresenta a proposta da construção de ApenMP, uma ferramenta para programação em arquiteturas multiprocessadas. Esta ferramenta deve oferecer uma interface de programação baseada no padrão OpenMP associada a um mecanismo de suporte à execução baseado em um escalonamento de lista. O projeto de ApenMP considera o modelo de programação e execução proposto por Anahy e as características de programação em OpenMP.*

1. Introdução

Um ambiente de programação deve oferecer recursos que viabilizem explorar eficientemente os recursos de hardware disponíveis sem onerar o processo de desenvolvimento de software. Desta forma, as abstrações de programação devem ser suficientes tanto para descrever uma determinada aplicação em termos de um programa como para determinar como este programa deve ser executado sobre os recursos disponibilizados em uma arquitetura. Considerando ambientes de programação para arquiteturas paralelas, a preocupação com o nível de abstração oferecido é ainda maior em função do aumento do número de recursos computacionais disponíveis.

Em arquiteturas paralelas, ferramentas de programação empregando modelos de programação baseados em multithreading (multiprogramação leve) são bastante populares. Em tais ferramentas, a presença de uma memória compartilhada permite que muitas das abstrações de programação, em particular o compartilhamento de dados via posições de memória, sejam baseadas no modelo imperativo seqüencial tradicional. Pthreads (POSIX threads) e OpenMP são exemplos desta classe de ferramentas [Akhter, S. and Robert, J. 2006].

O Projeto Anahy propõe um modelo para um ambiente de programação paralela de alto desempenho em arquiteturas paralelas. Este modelo é composto por duas camadas, uma definindo uma interface de programação de alto nível e outra um mecanismo de escalonamento de tarefas. Com estas duas camadas, o modelo permite que a concorrência de uma aplicação possa ser especificada sem considerar o paralelismo real oferecido pela arquitetura. Desse modo, é possível a adaptação da execução de programas para diferentes tipos de arquiteturas ajustando o algoritmo utilizado no balanceamento de carga, sem intervir no código da aplicação.

O modelo Anahy encontra-se implementado na ferramenta Athreads [Cavalheiro et al 2006] voltada para arquiteturas multiprocessadas e aglomerados de computadores. Athreads oferece uma interface de programação baseada em Pthreads e permite a descrição da concorrência de uma aplicação segundo o modelo de decomposição de tarefas [Cavalheiro 2004; Leopold 2001].

Este trabalho apresenta a proposta de desenvolvimento de ApenMP, uma ferramenta de programação implementando o modelo de execução proposto por Anahy mas com uma interface de programação baseada na especificação de OpenMP. No caso de ApenMP, será suportado o modelo de decomposição de dados, respeitando a natureza da interface de programação proposta por OpenMP.

Na continuação, a Seção 2 discute recursos de programação disponíveis em Athreads e em OpenMP e a Seção 3 apresenta o mecanismo de escalonamento de tarefas proposto por Anahy. A Seção 4 discute como um subconjunto de recursos de programação OpenMP devem ser implementados para dar suporte ao modelo Anahy .

2. Recursos de programação

A interface de programação de Athreads é baseada no modelo *fork/join* para descrever a concorrência em termos de threads. Athreads permite criação e sincronização de threads e compartilhamento de um espaço de endereçamento e foi implementada como uma biblioteca para programas em C/C++. Os principais recursos de programação oferecidos são *athread_create* e *athread_join,* para criação e sincronização de athreads. Os protótipos destes serviços são:

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

cujos parâmetros possuem a semântica definida pelo padrão POSIX: *func* é a função que deve ser executada pelo thread; *attr* especifica os atributos do thread que devem ser aplicados ao novo thread; *th* é um valor que será atualizado para identificar o thread criada; *in* é o endereço onde os dados de entrada da função definida por *func* irão ser armazenados. No serviço athread_join: *th* indica o thread no qual a sincronização irá ser realizada; *res* irá ser atualizada para apontar à posição na memória compartilhada onde os resultados da função executada pelo thread *th* podem ser encontrados.

A exemplo do que ocorre com o uso de ferramentas baseadas em Pthreads, a concorrência de um programa pode ser descrita por um grafo não estruturado de threads (Figura 1.a), uma vez que qualquer thread pode sincronizar (realizar uma operação de join) sobre qualquer outro thread por ele conhecido. No entanto, Athreads extende a especificação original permitindo que threads sofram múltiplas sincronizações por join. Outra extensão a implementação do esqueleto *split*/*compute*/*merge.* A operação *split* permite criar múltiplos fluxos de execução (réplicas) associados a um mesmo trecho de código. Analogamente, a operação *merge* permite sincronizar o término de todas as réplicas criadas. O compute, é de fato, o código do usuário a ser executado. As operações split e merge são encapsuladas pelos serviços *athread_create* e *athread_join*. Atributos do thread permitem especificar o número de réplicas a serem criadas (*splitfactor*) e também como o dado de entrada informado no *athread_create* deve ser particionado entre as réplicas e, de forma simétrica, como os dados de retorno das réplicas devem compor o resultado final à ocasião do *athread_join*.

A interface de programação do OpenMP também implementa o modelo *fork*/*join*, mas restringe a execução à forma de um grafo estruturado de threads (Figura 1.b). OpenMP permite a criação aninhada de threads e emprega diretivas de compilação como abstração para descrição da concorrência. Chamadas de serviços de bibliotecas permitem a interação do programa em execução com o ambiente de execução. As diretivas de compilação do OpenMP são compostas por sentinelas, diretivas e cláusulas.

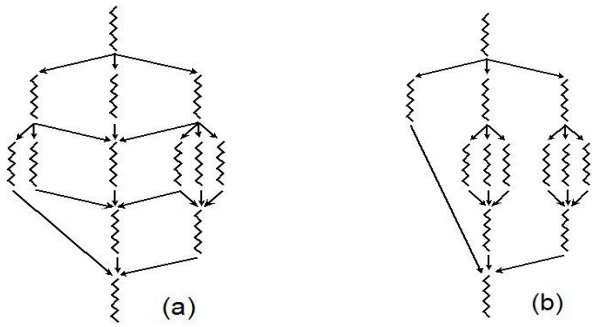

Figura 1. Grafo de dependência entre (a) tarefas em Anahy e (b) regiões paralelas em OpenMP.

Um exemplo de criação de uma região paralela em OpenMP é dado por:

```
x = 1, y = 2;
#pragma omp parallel private(x)
x=y+1;
```

Neste exemplo, a diretiva *parallel* implica na criação de threads no espaço usuário (regiões paralelas) que deverão ser executados pelo número de threads de serviço do ambiente de execução. Este número corresponde ao valor associado à variável de ambiente OMP_NUM_THREADS. No trecho de código é ilustrado a comunicação de dados entre os threads. O comportamento padrão implica que todos os dados locais ao thread original sejam acessíveis ao thread criado, como a variável *y*. Porém, a variável *x* na cláusula *private* indica que cópias locais de *x* devem ser instanciadas para uso local de cada thread.

3. Escalonamento

Anahy propõe o uso de algoritmos de escalonamento de listas. Uma camada entre os recursos de programação e o mecanismo de escalonamento é responsável pela criação de um grafo direcionado de tarefas acíclico (DAG) representando a execução do programa. Neste DAG cada vértice corresponde a uma tarefa a ser executada pelo programa e cada aresta dirigida uma comunicação de dados entre tarefas. O *caminho crítico* em um DAG representa o limite de tempo de execução do programa em uma arquitetura paralela [Graham et al 1972], sendo este o tempo necessário à execução da maior seqüência de dependências entre tarefas.

Em um DAG uma tarefa é definida por um conjunto de instruções que devem ser executadas de forma seqüencial e que recebe, como entrada, um conjunto de dados e fornece outro como saída. Uma tarefa está pronta para ser executada no momento em que sua dependência estiver satisfeita, ou seja, quando o conjunto de dados de entrada estiver disponível para leitura.

Em Athreads tarefas são encapsuladas no contexto de threads (fluxo de controle e pilha de execução) e um grafo representando as dependências entre as tarefas é mantido em tempo de execução. Este grafo é explorado segundo algoritmos de escalonamento de lista para otimização da execução [Cordeiro et al 2005].

Em OpenMP o escalonamento das regiões paralelas obedece a dependência definida pelo fluxo de execução do programa. É possível também o uso de estratégias oferecidas pelo padrão OpenMP para ajustar o particionamento e distribuição das regiões paralelas entre os threads de serviço. No entanto, OpenMP não aplica um mecanismo baseado em algoritmo de lista para controlar a execução do programas.

4. Conclusão e trabalhos futuros

A ferramenta de programação ApenMP deve implementar o modelo de execução Anahy utilizando como interface de programação um subconjunto da especificação de OpenMP. As regiões paralelas definidas pelo programa em execução comporão as tarefas requeridas pelas estratégias de escalonamento de lista, sendo as dependências entre estas representadas pelas precedências de execução identificada pela disposição das instruções no próprio código e pelos acessos realizados pelas regiões paralelas (*private*, *reduce*, *lastprivate*, *firstprivate*). O protótipo a ser construído deverá considerar, inicialmente, a implementação de suporte às diretivas *parallel* e *for*. Um exemplo de código é apresentado na Figura 2.

```
int v[100], i, somatorio;
InicializaVetor(v);
#pragma omp for private(i), reduction(+:somatorio)
  for( i = 0 ; i < 100 ; i++ ) soma += v[i];
ImprimeSoma(somatorio);
```

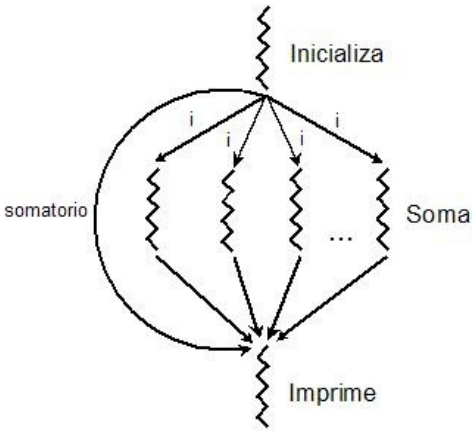

Figura 2. Exemplo de código em ApenMP e o grafo de tarefas gerado.

Espera-se, com esta implementação, validar o modelo Anahy com a utilização de uma feramenta de programação seguindo o padrão OpenMP. O protótipo poderá ser incrementado com a adição de outras funcionalidades suportadas por OpenMP.

Referências

Akhter, S. and Robert, J. (2006). Multi-core programming: Increasing performance through software multithreading. Intel Press. Hillsboro.

Cavalheiro, G. G. H. (2004) Princípios da programação concorrente. In ERAD 2004.

Cavalheiro, G. G. H., Gaspary, L. P., Cardozo, M. A., Cordeiro, O. C. (2006) Anahy: A programming environment for cluster computing. In: VECPAR 2006 (LNCS 4395).

Cordeiro, O. C., Peranconi, D. S., Villa Real, L. C., Dall'Agnol, E. C., Cavalheiro, G. G. H. Exploiting Multithreaded Programming on Cluster Architectures. In HPCS 2005.

Graham, L. R. (1969). Bounds on multiprocessing tming anomalies. In SIAM Journal of Applied Mathematics, v.(17):2.

Leopold, C. (2001). Parallel and distributed computing. John Wiley & Sons, England.