Tradução de código OpenMP para Athreads através de ApenMP

Cristian Fernando Flores Castañeda[1], Leonardo Lobo da Luz[1], Gerson Geraldo Homrich Cavalheiro[1]

[1]Instituto de Informática – Universidade Federal de Pelotas (UFPel)
Caixa Postal 354 – 96.010-900 – Pelotas – RS – Brasil

{cristian.castaneda, gersonc}@anahy.org, loboluz161@gmail.com

Resumo. *Este trabalho apresenta o pré-processador ApenMP, uma ferramenta para programação em arquiteturas multiprocessadas capaz de extrair especificações de paralelismo de dados e utilizar escalonamento baseado em algoritmos de listas para obter o desempenho máximo da arquitetura. O projeto de ApenMP considera o modelo de programação e execução proposto por Anahy e as características de programação de OpenMP. O desempenho obtido por esta ferramenta é comparado com os oferecidos por compiladores OpenMP e Pthreads.*

1. Introdução

Atualmente, o processamento paralelo é essencial em áreas onde as aplicações demandam altos recursos de processamento. A popularização de configurações para máquinas multiprocessadas, devido ao barateamento de processadores *multi-core*, ocasionou uma alta demanda por aplicações que incorporem o modelo da multiprogramação leve, o qual permite que diversos fluxos de execuções (*threads*) estejam ativos durante a execução do programa [Cavalheiro 2004].

Diversas ferramentas comerciais oferecem recursos para explorar este modelo de programação. Entre as mais populares estão Pthreads e OpenMP [Akhter and Robert 2006]. Enquanto Pthreads provê uma interface para destacar o paralelismo de tarefas da aplicação, OpenMP provê uma interface voltada ao paralelismo de dados. No meio acadêmico, Anahy destaca-se por oferecer estratégias de escalonamento de tarefas que permitam obter o máximo desempenho de uma arquitetura paralela pela aplicação de algoritmos de listas [Cavalheiro 2004].

Este trabalho apresenta a ferramenta de pré-processamento ApenMP, permitindo a execução de programas utilizando o modelo de execução de Anahy, mas com especificações de programação baseadas no modelo OpenMP. Uma análise de desempenho permitiu verificar que o desempenho de programas ApenMP reflete o comportamento obtido para programas Athreads, uma implementação baseada em Pthreads do modelo Anahy [Cavalheiro et al. 2006].

Na continuação, nas Seções 2 e 3 são apresentados os modelos e recursos de programação e de escalonamento de OpenMP e Anahy, respectivamente. Na Seção 4 é descrita a ferramenta de pré-processamento ApenMP. Na Seção 5 são discutidos resultados de desempenho de ApenMP. Finalmente na Seção 6, são apresentadas conclusões sobre o projeto e propostas para trabalhos futuros.

2. OpenMP

O modelo de programação de OpenMP baseia-se no modelo *fork/join* para criação e sincronização de *threads*. A aplicação, quando em execução, dispara trechos de códigos que são executados em regiões paralelas pelos *threads* de serviços. Estes *threads* de suporte à execução são reunidos em um núcleo denominado *thread-pool*. A interface de programação de OpenMP possui diretivas de compilação como abstração para criação e sincronização de *threads* e chamadas a serviços de biblioteca para permitir a interação do programa em execução com o seu ambiente [Chandra et al. 2001]. O protótipo básico de uma região paralela em OpenMP é ilustrada na Fig. 1.

```
#pragma omp diretiva [clausula]
  corpo
```

Figura 1. Protótipo de uma região paralela em OpenMP.

Nesta sintaxe, *diretiva* pode ser *parallel*, a qual indica a criação de trechos de códigos concorrentes, e *for* que indica a paralelização de um laço. Em *clausula* é identificada a natureza dos dados manipulados [Chandra et al. 2001].

Em relação ao escalonamento, este pode ser *Static*, *Dynamic*, *Guided* e *Runtime*. Estes escalonamentos identificam políticas próprias em relação à distribuição de *chunks* (conjuntos de dados) entre os *threads* do mesmo *thread-pool* [Chandra et al. 2001].

3. Anahy

O ambiente de execução Anahy propõe um modelo de programação composto por uma camada definindo uma interface de programação de alto nível e por outra camada definindo 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. Na Fig. 2 é ilustrado o processo de criação e sincronização de *threads*.

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

Figura 2. Criação e sincronização de *threads* em Anahy.

No serviço *athread_create()*, o parâmetro *func* indica a função que deve ser executada pelo *thread*; *attr* especifica os atributos do *thread*; *th* é um valor que será atualizado para identificar o *thread* criado; *in* é o endereço onde os dados de entrada serão armazenados. No serviço *athread_join()*, *th* indica o *thread* no qual a sincronização será realizada; *res* aponta para uma posição na memória onde os resultados da função executada pelo *thread th* serão encontrados.

Em relação ao escalonamento, Anahy utiliza algoritmos de listas [Cordeiro et al. 2005]. 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. Este grafo é submetido ao núcleo executivo para escalonar as tarefas em execução sobre os recursos computacionais disponíveis [Cavalheiro et al. 2006].

4. ApenMP

A ferramenta de programação ApenMP foi implementada em linguagem C e oferece suporte às diretivas *parallel* e *for*, e também às cláusulas *private*, *firstprivate*, *lastprivate* e *reduction*. As ferramentas utilizadas para a criação de ApenMP foram *Flex* e *Bison*.

O pré-processador de ApenMP deve receber como entrada um código escrito segundo a especificação OpenMP. Cabe a este pré-processador analisar os resultados de pré-processamento no código original e realizar as transformações necessárias para a geração de código Athreads. A tradução consiste em identificar no código definições das cláusulas dos dados locais aos *pragmas* e identificar as diretivas OpenMP (*parallel* e *for*). O programa tem então seu conteúdo expresso em OpenMP traduzido em seu equivalente em Athreads. Na Fig. 3, é ilustrado o código de entrada para ApenMP.

```
#include <omp.h>
void main(int argc, char **argv){
     int x, y, soma;
     #pragma omp parallel private(x, y, soma)
             soma = x + y;
}
```

Figura 3. Código original em OpenMP.

Na Fig. 4, é apresentado o código de saída obtido através de ApenMP

```
#include "athread.h"
void main(int argc, char **argv){
     int x, y, soma, ampAux, numThreads;
     athread_t ampth[numThreads];
     struct valores *head;
     void *retorno;
     aInit(&argc, &argv);
     for(ampAux = 0; ampAux < numThreads; ampAux++)
        athread_create(&ampth[ampAux], NULL, amp1, head);
     for(ampAux = 0; ampAux < numThreads; ampAux++)
        athread_join(ampth[ampAux], &retorno);
     aTerminate();
}

void amp1(void *dados){
     struct valores *res;
     int x, y, soma;
     soma = x + y;
     return (void *) res;
}
```

Figura 4. Código traduzido para Athreads.

5. Resultados de Desempenho

A obtenção dos resultados de desempenho foi realizada utilizando um computador com processador *Intel Core* 2 *Duo* com freqüência de *clock* 1,86GHz, memória *cache* de 4MB e memória RAM de 2GB. O Sistema Operacional utilizado foi o *GNU-Linux kernel* 2.6.24, compilador Intel e GCC 4.2.4 para compilação de aplicações OpenMP.

A metodologia de avaliação de desempenho foi contabilizar o tempo de execução de um programa em segundos. O tempo de execução somente contabiliza o tempo das regiões paralelas. Os tempos apresentados representam uma média e desvio padrão de 15 execuções e para cada caso foram utilizados 2, 4 e 8 *threads*. Foram

escolhidas duas abordagens de paralelismo, tarefas e dados para avaliar o desempenho de execução final de Anahy em relação à OpenMP e Pthreads através das duas abordagens. Desse modo, foi utilizada uma aplicação para cada abordagem, sendo que, para cada aplicação, foram obtidas 3 versões: OpenMP, ApenMP (versão em Athreads) e Pthreads.

Na Tab. 1, é mostrado o resultado de desempenho utilizando uma aplicação que especifique o paralelismo de tarefas. Como pode ser observado, o tempo de execução da versão através ApenMP foi inferior aos demais. Este resultado é satisfatório, mas cabe ressaltar que o paralelismo de tarefas não é o nicho de OpenMP.

Tabela 1. Tempo de execução das aplicações caracterizando paralelismo de tarefas.

	OpenMP(Intel)			OpenMP(GCC)			ApenMP			ApenMP→Pthreads		
Threads	2	4	8	2	4	8	2	4	8	2	4	8
Média	0,155	0,260	0,438	0,159	0,262	0,441	0,141	0,248	0,410	0,167	0,268	0,574
Desvio padrão	0,009	0,001	0,003	0,011	0,002	0,137	0,001	0,008	0,122	0,002	0,022	0,089

Por outro lado, na Tab. 2 é ilustrado o desempenho obtido utilizando uma aplicação que especifica o paralelismo de dados. Nesta abordagem, o tempo OpenMP foi inferior aos demais. Porém, o tempo de ApenMP foi menor do que Pthreads.

Tabela 2. Tempo de execução das aplicações caracterizando paralelismo de dados.

	OpenMP(Intel)			OpenMP(GCC)			ApenMP			ApenMP→Pthreads		
Threads	2	4	8	2	4	8	2	4	8	2	4	8
Média	0,101	0,103	0,111	0,101	0,105	0,115	0,187	0,211	0,356	0,196	0,236	0,369
Desvio padrão	0,001	0,003	0,011	0,002	0,006	0,011	0,056	0,012	0,013	0,031	0,004	0,006

6. Conclusões e Trabalhos Futuros

Com uso de ApenMP, o programador pode se beneficiar das vantagens da simplicidade de paralelização de um programa escrito em OpenMP e obter um desempenho eficiente pela utilização de escalonamento de algoritmos de listas provido por Athreads. Os resultados de desempenhos foram satisfatórios, refletindo a abordagem de paralelismo de cada experimento. Como trabalhos futuros, espera-se adicionar novas funções de OpenMP e um núcleo executivo próprio, eliminando a geração de código intermediário.

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., Cardoso, M. A., Cordeiro, O. C. (2006) “Anahy: A Programming Environment for Cluster Computing”. In VECPAR 2006 (LNCS 4395).

Chandra, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J., and Menon, R. (2001) “Parallel Programming in OpenMP”. Morgan Kaufmann, San Francisco.

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