Desenvolvimento de aplicações para teste e avaliação de desempenho de Athread

Lucas G. Nachtigall[1], **Alan Schlindvein de Araújo**[1],
Douglas Eduardo Rosa[1], **Gerson Geraldo Homrich Cavalheiro**[1]

[1]Centro de Desenvolvimento Tecnológico – Ciência da Computação
Universidade Federal de Pelotas (UFPel)
Loboratory of Ubiquitous and Parallel Systems - LUPS
Caixa Postal 354 – 96.001-970 – Pelotas – RS – Brazil

{lgnachtigall,asdaraujo,derosa,gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Com a evolução dos processadores multi-cores e com o aumento do uso de programas usando processamento paralelo, é necessário o desenvolvimento e evolução dos ambientes de programação de computação paralela. Este trabalho documenta o desenvolvimento de um conjunto de programas para testes e avaliação de desempenho desenvolvidos para o ambiente de programação concorrente Anahy. Versões destes programas também foram desenvolvidos para outras ferramentas de programação, a fim de permitir uma futura comparação de desempenho destas com o obtido por Anahy.*

1. Introdução

Este trabalho está sendo realizado no contexto do projeto Anahy, que tem por objetivo o desenvolvimento de um ambiente de programação concorrente sobre arquiteturas multiprocessadas. Athread [Cavalheiro 2006] implementa o modelo de programação e execução Anahy sobre arquiteturas multiprocessadadas. O objetivo pontual da presente atividade é desenvolver um conjunto de aplicações para Athread, de forma que estas aplicações permitam realizar testes de robustez e avaliação de desempenho de Athread.

Em complementação ao objetivo principal, versões das aplicações desenvolvidas para Athreads foram também desenvolvidas para outras ferramentas de programação *multithread*. O desenvolvimento destas versões permite tanto realizar uma análise da aplicabilidade da interface de programação proposta por Athread frente as interfaces de programação apresentadas pelas demais ferramentas como também a comparação dos desempenhos obtidos.

O restante do documento apresenta, na Seção 2, Athread e os demais ambientes de programação *multithreaded* utilizados: Cilk, Kaapi e Pthreads. A Seção 3 apresenta as aplicações desenvolvidas: cálculo da série de Fibonacci, ordenamento usando Mergesort, método de Monte-Carlo, algorítmo de multiplicação de matrizes e algorítmo das n-queens. A análise comparativa entre Athreads e as demais ferramentas não é tratada neste trabalho.

2. Ambientes utilizados

2.1. Athread

O Athread é a ferramenta de principal foco deste artigo, é um ambiente de alta performance de computação paralela que funciona sobre arquiteturas SMP e em *clusters* de

computadores e implementa o modelo de *threads* M:N. Uma de suas características mais importantes é sua interface de programação que permite uma utilização mais simples em relação ao uso de ferramentas básicas [Cavalheiro 2006].

Esse objetivo é atingido utilizando mecanismos de escalonamento que ocultam as características físicas do hardware, fazendo que o programador possa descrever a concorrência de seus programas sem precisar definir como os recursos da arquitetura vão ser explorados.

A interface mencionada oferece duas primitivas para manipulação de *threads* e comunicação de resultados: *create* e *join*. O resultado disso é ter o programa descrito em termos de grafos dirigidos com ciclos (DCGs) [Araujo 2009].

2.2. Cilk

Cilk é uma linguagem para programação *multithread* paralela baseada no ANSI C, ela foi designada para programação paralela de propósito geral, mas é especialmente efetiva para a exploração do paralelismo dinâmico e altamente assíncrono, que em algumas vezes pode ser difícil de ser escrito em dados paralelos ou no estilo de passagem de mensagens. Um programa Cilk utiliza o parâmetro *spawn* para a criação de threads e o parâmetro *sync* para a sincronização de procedimentos.

2.3. Kaapi

Kaapi é um *middleware* que trabalha em grafos de fluxos de dados acíclicos e dinâmicos. Depois de ter esse grafo, pode ser feito o escalonamento dinamicamente usando algorítmos de *work-stealing*.

O Kaapi usa o paradigma de descrever e analisar a tarefa do grafo ou programa e descrever as dependências entre as tarefas, depois que isso é feito o Kaapi escalona as tarefas de uma maneira eficiente onde todas as dependências são respeitadas e o paralelismo entre tarefas independentes é usado ao máximo possível. A criação de novas threads é feita utilizando parâmetro *fork* e *sync* para sincronização.

2.4. Pthreads

Com a padronização das interfaces de programação paralela, foi criado o padrão POSIX para sistemas UNIX. Interfaces que fazem parte desse padrão foram chamadas de POSIX *threads*, ou Pthreads.

Pthreads são um conjunto de procedimentos e tipos escritos na linguagem em C, implementados com o arquivo de include/cabeçalho ”pthread.h”. A criação de novas *threads* é feita utilizando o *pthread_create()* e a sincronização é feita pela função *pthread_join()*.

3. Benchmarks

3.1. Fibonacci

A função recursiva abaixo demonstra o funcionamento da série de Fibonacci:

$$F(n) = \begin{cases} 0, & se\ n = 0; \\ 1, & se\ n = 1; \\ F(n-1) + F(n-2) & outros\ casos. \end{cases}$$

A série começa com os números 0 e 1, e a partir disso obtêm-se o próximo número da série somando-se os 2 anteriores e assim sucessivamente. Esse algorítmo foi escolhido pela facilidade de demonstração do paralelismo onde a cada vez que vai ser calculado um número $F(n)$ onde $n > 1$, vão ser criadas duas linhas de execuções, uma para $F(n-1)$ e outra para $F(n-2)$ que após serem executadas, vão ser sincronizadas e somadas para formar o resultado.

3.2. MergeSort

O MergeSort é um algorítmo de ordenação do tipo "dividir para conquistar", onde sua finalidade é dividir uma sequência original em sequências menores até que se formem pares de dados, esses pares são ordenados com um outro par formando sequências de 4 dados, e assim sucessivamente.

Pode-se criar múltiplas threads onde cada thread irá dividir a sequência de dados que lhe foi passada, até se obterem os pares de dados. Então é feita a sincronização das linhas de execução de modo que quando for necessária a ordenação com outras divisões, ambas as linhas já tenham terminado sua ordenação.

3.3. Multiplicação de matrizes

Na multiplicação de matrizes são realizadas multiplicações das colunas de uma matriz A pelas linhas de uma matriz B, de modo que o produto AB é dado por:

$$(AB)[i,j] = A[i,1]B[1,j] + A[i,2]B[2,j] + ... + A[i,n]B[n,j]$$

Em uma execução sequencial, seria necessário fazer n iterações (onde n é o número de linhas) sendo que cada iteração multiplica uma linha da matriz A pelas colunas da matriz B. No algorítmo implementado é criado uma linha de execução para um número indicado de linhas, então cada linha de execução executa paralelamente um número k de iterações informada pelo usuário.

3.4. Método de Monte-Carlo

Nessa implementação o método tem como objetivo calcular o número π, usando a formula $\pi = (4 \times m)/n$, se um círculo de raio R está inserido dentro de um quadrado de tamanho $2R$, então a área do círculo vai ser $\pi \times R^2$ e a área do quadrado vai ser $(2R)^2$. Assim a relação da área do círculo com a área do quadrado será de $\pi/4$. Então se forem selecionados n pontos dentro do quadrado, $n \times (\pi/4)$ desses pontos deveriam estar dentro do circulo.

Se tomarmos m como o número de pontos que estão dentro do circulo, temos que $n \times (\pi/4) = m$. E dessa formula pode-se deduzir que $\pi = (4 \times m)/n$.

Para a execução do algorítmo, são criadas k linhas de execução onde cada uma delas vai receber uma porção de n, e calcular m. Depois que todas as linhas terminarem suas execuções, é somado os valores de m de cada uma das linhas de execução e colocadas na fórmula já mencionada para chegar no resultado aproximado de π.

3.5. n-queens

O algorítmo de n-queens consiste em um tabuleiro $n \times n$ onde são colocadas n rainhas de modo que nenhuma ataque a outra. No xadrez, a rainha é uma peça que ataca de qualquer distância nos sentidos vertical, horizontal e diagonal.

De uma forma sequencial, o algorítimo começa posicionando a primeira rainha na primeira coluna e após é usado o método de *backtrack*, ou seja, a próxima rainha é colocada em uma coluna em que não vai causar conflito, se nenhuma coluna for encontrada os passos são revertidos até uma situação válida e então se escolhe uma coluna diferente.

Para a paralelização desde algorítmo, a escolha da próxima coluna disponível é feita paralelamente, aumentando o desempenho e velocidade do algorítmo.

3.6. Torre de Hanoi

Usando um algorítmo recursivo, a ideia é levar o maior disco para o último pino, mas para isso todos os outros discos tem que serem retirados do primeiro pino e colocados no pino do meio, assim, logo após o maior pino ter sido passado, começa o mesmo algorítmo para passar o segundo maior. Com isso os discos são passados entre os pinos de origem, destino e auxiliar até que seja possível mover o disco maior até o pino desejado.

Paralelamente, esse algorítmo é implementado de modo que em cada chamada recursiva, vai ser criada uma nova thread.

4. Conclusão

Este trabalho apresentou um conjunto de aplicações desenvolvidas para testar o ambiente de programação Athread e avaliar seu desempenho. Versões destas aplicações também foram desenvolvidas para outras ferramentas de programação *multithread*. Os programas desenvolvidos para Athread foram incorporados no pacote de distribuição da ferramenta com o intuito fornecer exemplos de utilização.

Do trabalho realizado até o momento, destaca-se que as diversas ferramentas utilizadas possuem características próprias de programação, embora todas respondam ao modelo *multithread*. Em Athreads o destaque se dá para uma interface próxima a um padrão já estabelecido, que é o de Pthread, associado a um núcleo de escalonamento aplicativo. Pthread também foi utilizada neste trabalho, mas destaca-se que esta ferramenta não foi desenvolvida focando processamento de alto desempenho. As demais, Cilk e Kaapi, possuem núcleos de escalonamento aplicativo, possuindo, no entanto, interfaces de programação próprias.

Na sequência do trabalho será realizada a coleta de resultados de desempenho dos programas desenvolvidos sobre as diferentes ferramentas de forma a melhor compreender o desempenho do núcleo de escalonamento implementado em Athreads.

Referências

[Araujo 2009] Araujo, A. S. (2009). Utilização de afinidade no escalonamento de threads em multiprocessadores. In *IX Escola Regional de Alto Desempenho.* Caxias do Sul, 2009.

[Cavalheiro 2006] Cavalheiro, G. G. H. ; Gaspary, L. P. (2006). Anahy: A Programming Environment for Cluster Computing. *High Performance Computing for Computational Science - VecPar 2006, 2007, Rio de Janeiro. LNCS High Performance Computing for Computational Science - VecPar 2006. Heildelberg : Springer-Velag, 2007*, pages 198–211.