Um estudo de caso de paralelização de um algoritmo concorrente recursivo em ferramentas de programação multithread

Guilherme P. Britto Cousin, Rodolfo M. Favaretto, Gerson Geraldo H. Cavalheiro

[1]Centro de Desenvolvimento Tecnológico – CDTec
Universidade Federal de Pelotas – UFPel
Pelotas, RS – Brasil

{gpbcousin, rmfavaretto, gerson.cavalheiro}@inf.ufpel.edu.br

Resumo. *Este trabalho tem como objetivo uma revisão das ferramentas Cilk Plus e OpenMP, apontando as estratégias de execução e também analisa o conjunto de recursos para a criação e sincronização das tarefas nestas ferramentas. Sendo possível observar que quanto maior o conhecimento do programador sobre a ferramenta, melhor proveito é tirado dos recursos disponíveis.*

1. Introdução

De acordo com a constante evolução de hardware paralelo, e para explorar a fundo os seus recursos, a implementação de software paralelo tem avançado significativamente. Isso ocorre pois só é possível explorar eficientemente estas arquiteturas quando é evitado desperdício de tempo de processamento por ociosidade do processador. Ciclos ociosos de processamento em uma arquitetura multiprocessada podem ser consequência da pouca experiência do programador quando da distribuição do custo computacional de seu programa entre as unidades de processamento disponíveis. O desejável para aumentar a produtividade de software e sua robustez é incluir mecanismos de exploração dos recursos de processamento como uma abstração da própria linguagem de programação, provendo uma abstração ao problema de escalonamento das atividades paralelas [Skillicorn and Talia 1998].

O escalonamento tem como função promover a execução das atividades geradas pelo programa. Caso o escalonador também possua a responsabilidade de realizar o balanceamento de carga, deve garantir, segundo alguma heurística particular, alguns índices de desempenho na execução da aplicação. Existem diversas ferramentas de programação multithread [Cavalheiro and Bois 2014], entre elas OpenMP [Chandra 2001] e Cilk [Blumofe et al. 1995], que oferecem recursos de escalonamento. Embora o escalonamento provido por estas ferramentas seja capaz de explorar, segundo uma heurística, eficientemente o hardware disponível, para que não haja comprometimento do desempenho, os programas devem ser implementados com os recursos oferecidos por suas respectivas interfaces de programação (APIs) de forma a refletir a heurística de escalonamento adotada.

As interfaces de programação disponibilizadas pelas ferramentas oferecem diferentes alternativas para desenvolvimento dos programas. Quanto maior o número de alternativas, maior liberdade o programador tem para descrever a concorrência de seu

programa, dificultando as decisões de escalonamento. Este trabalho tem como objetivo estudar este problema, comparando as diferentes alternativas de programação oferecidos por OpenMP e Cilk Plus em relação ao desempenho obtido. Este artigo esta organizado da seguinte maneira. A seção 2, apresenta as interfaces de programação de Cilk e OpenMp. Na seção 3, é descrito um exemplo e a utilização de das interfaces descritas na seção 2.

2. Interfaces de programação

O caso de estudo documentado neste texto restringe a análise ao conjunto de recursos para criação e sincronização de tarefas em OpenMP e Cilk Plus. Também é disticutido um único tipo de problema, caracterizado por um algorítmo concorrente recursivo desbalanceado, representado por um algoritmo de Fibonacci. Outras funcionalidades, como operações atômicas, mecanismos de sincronização e operações de redução de dados também não são objetos de análise neste texto.

Cilk Plus é uma ferramenta de programação altamente especializada para execução de algoritmos concorrentes recursivos. Sua API é minimalista, contando apenas com uma operação de `spawn` para criar threads e uma operação `sync` para sincronizar o thread corrente com o término de todos os threads que tenha algoritmo crido. Apesar de extremamente enxuta, o uso de forma inadequada destas primitivas, não refletindo a heurística de escalonamento adotada, impacta significativamente no desempenho final. A Figura 1(a) apresenta o código de uma estratégia recursiva do cálculo de Fibonacci em Cilk Plus. Onde é chamado a diretiva `cilk_spawn` na linha 9 são criadas *threads* dando preferência a execução do caminho crítico da aplicação, e armazendo a continuação do *thread* criador na linha 10. Na linha 11 é feita a sincronização das threads.

OpenMP oferece diversas possibilidades para desenvolvimento de código paralelo para um mesmo programa. A Figura 1(b) apresenta uma implementação recursiva do cálculo de Fibonacci com a utilização de `task`, onde tarefas são criadas de forma explícita (linhas 9 e 11). A sincronização destas tarefas é realizado com `taskwait` (linha 13). A Figura 1(c) apresenta a criação de tarefas com a `sections` (linha 9) delimitando o código de cada tarefa em diferentes `sections`. Estas duas formas de descrição de concorrência ativam distintos mecanismos de escalonamento em OpenMP. No primeiro caso, a estratégia explora a estrutura recursiva aninhada do problema, na segunda não pois não é criado um relacionamento entre as tarefas.

```
1  long fib_cilk(int n) {
2    long x, y;
3    if (n < 2){
4      return n;
5    }
6    if (n < threshold){
7      return seq_fib(n);
8    }
9    x = cilk_spawn fib_cilk(n-1);
10   y = fib_cilk(n-2);
11   cilk_sync;
12   return x + y;
13 }
```

(a)

```
1  long fib_openmp(int n){
2    long fnm1, fnm2, fn;
3    if ( n < 2 ){
4      return(n);
5    }
6    if (n < threshold){
7      return seq_fib(n);
8    }
9    #pragma omp task shared(fnm1)
10      fnm1 = fib_openmp(n-1);
11   #pragma omp task shared(fnm2)
12     fnm2 = fib_openmp(n-2);
13   #pragma omp taskwait
14     fn = fnm1 + fnm2;
15   return(fn);
16 }
```

(b)

```
1  long fib_openmp2(int n){
2    long fnm1, fnm2, fn = 0;
3    if ( n < 2 )
4      return(n);
5    if (n < threshold)
6      return seq_fib(n);
7    #pragma omp parallel
8    {
9      #pragma omp sections
10     {
11       #pragma omp section
12         fnm1 = fib_openmp2(n-1);
13       #pragma omp section
14         fnm2 = fib_openmp2(n-2);
15     }
16   }
17   return(fnm1+fnm2);
18 }
```

(c)

Figura 1. Exemplos de códigos utilizados na análise.

3. Resultados e Discussões

Com a interface provida por estas ferramentas, diversas implementações podem ser realizado para o algoritmo proposto. O desempenho, em termos de tempo de execução, estão apresentados na Figura 2. As curvas Cilk1, Omp1, Omp3 correspondem as implementações dos códigos da Figura 1.a, b e c, respectivamente. As demais curvas, variações sobre estes casos base.

A curva Cilk2 representa uma implementação em que a computação de Fibo(n-2) é realizado por um segundo thread, havendo um `spawn` adicional na linha 10. Neste caso, a distância em relação à Cilk1 corresponde ao overhead de criação deste novo thread. A implementação Cilk3 é uma variação do código da Figura 1.a em que a criação do thread é feita para o cálculo de Fibo(n-2) e Fibo(n-1) executa sobre o corpo do thread corrente. A distância de Cilk3 em relação à Cilk1 reflete o uso dos recurso de programação de forma a não refletir a estratégia de escalonamento da ferramenta, a qual prioriza a execução do caminho crítico.

A curva Omp2 representa uma variação do código da Figura 1.b onde não é criada uma nova tarefa para execução de Fibo(n-2). Em relação ao desempenho apresentado pela curva Omp1, observa-se redução no tempo de execução pela diminuição do sobrecusto de criação de tarefas. A curva Omp3(a) deve ser comparada à Omp3, uma vez que seu código é uma alternativa ao código da Figura 1.c utilizando um mecanismo de redução de dados compartilhados para obter o resultado final, não sendo observada diferença significativa no uso deste recurso. A curva Omp4 reflete o desempenho de uma implementação equivalente ao código da Figura 1.b onde a tarefa para cálculo de Fibo(n-2) é executada antes da criação da tarefa Fibo(n-1). Comparados os tempos de execução entre as curvas Omp1 e Omp4, a segunda apresenta um melhor desempenho, indicando que a priorização das tarefas não segue a mesma estratégia de Cilk Plus.

Os resultados apresentados foram executados em computador, em uso dedicado, dotado com um processador Intel Core i7-2600 CPU @ 3.40GHz (4 núcleos com SMT ativo), com memória de 8GiB. Os resultados estão apresentados em forma de gráfico que mostra a média de 30 execuções, foram feitos testes estatísticos para a validação destes resultados.

4. Conclusão

Cilk Plus e OpenMP são ferramentas de programação multithread que incorporam estratégias de escalonamento em nível aplicativo. Estas estratégias de escalonamento tem o objetivo de reduzir o tempo total de execução do programa. O escalonador de Cilk Plus privilegia a execução do caminho crítico do programa. Como heurística, assume que o programa possui uma estrutura recursiva aninhada, sendo que as tarefas mais antigas são as que possuem maior quantidade de trabalho. O escalonamento em OpenMP, por sua vez, possui dois comportamentos, um quando as tarefas são geradas de forma implícita, como as geradas pelas diretivas `for` e `section`, e outro quando são gerados explícitamente pela diretiva `task`. No primeiro caso, todas as tarefas são consideradas idênticas entre si. No segundo, considera-se a natureza aninhada e recursiva da aplicação, sendo privilegiada a exploração de localidade das tarefas.

Neste trabalho foi destacado o impacto que diferentes usos da API de uma ferramenta implica no desempenho de um programa. A continuação deste trabalho será a

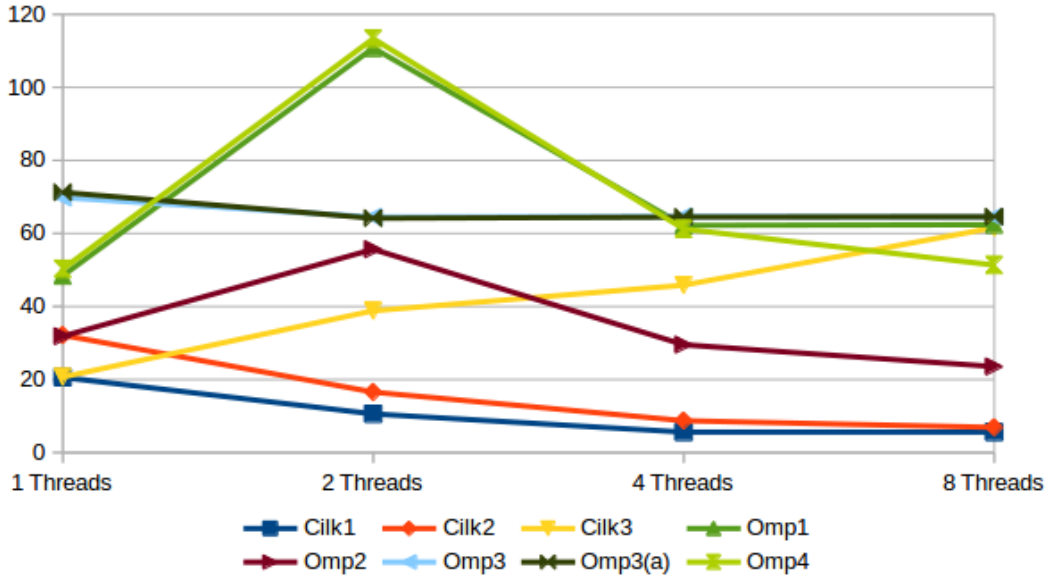

Figura 2. Tempo em segundos das execuções da posição 45 da serie de fibonacci, com threshold na posição 10

completa análise dos recursos oferecidos por diferentes ferramentas de programação multithread para um amplo leque de aplicações.

Referências

Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H., and Zhou, Y. (1995). Cilk: an efficient multithreaded runtime system. In *Proc. Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*.

Cavalheiro, G. G. H. and Bois, A. D. (2014). *Atualizações em Informática 2014*, chapter Ferramentas Modernas para Programação Multithread, pages 49–91. Sociedade Brasileira de Computação, Porto Alegre/RS.

Chandra, R. (2001). *Parallel programming in OpenMP*. Morgan Kaufmann.

Skillicorn, D. B. and Talia, D. (1998). Models and languages for parallel computation. *ACM Computing Surveys (CSUR)*, 30(2):123–169.