Paralelização de Algoritmos utilizando a biblioteca OpenMP

Carlos Alberto Franco Maron[1]

[1] Programa de Pós-Graduação em Ciência da Computação
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

carlos.maron@acad.pucrs.br

Resumo. *Este trabalho apresenta resultados da implementação de algoritmos paralelizados utilizados a biblioteca de programação OpenMP. No trabalho, possui um breve resumo da implementação destes algoritmos, e uma comparação dos tipos de escalonadores, onde dynamic e guided tiveram um resultado positivo.*

1. Implementação

O código para multiplicação de matrizes foi desenvolvido utilizando as sintaxes de programação da linguagem C. Com isso, a iniciação do código sequencial tem como parâmetro de entrada o valor que será o tamanho da matriz à ser calculada pelo código, e subsequente a isso solicitado como parâmetro um número de repetições, que posterioremente é utilizado para um cálculo de média do tempo de execução. No código paralelo, o algoritmo recebe os mesmos parâmentros, porém, é solicitado o número de *threads* que se deseja executar.

O algoritmo matemático escolhido para ser paralelizado tem como objetivo encontrar todos os números primos dos números 5 até 500.000 multiplicados por 10, ou seja, o código é iniciado em 5, depois 50, 500, 5.000, 50.000 e por fim 500.000. O código sequencial ao ser iniciado não solicita nenhum parâmetro ao usuário, porém o paralelo é necessário setar a variável de ambiente que passa ao código o número de *threads* para ser executado. Esse procedimento, é feito inserindo a seguinte linha de comando no terminal do linux: **export OMP_NUM_THREADS=***. Neste caso, o símbolo asterisco é substituído pelo quantidade de *threads* que seu código será executado. Todos os códigos foram executados utilizando um nodo no *cluster* Atlântica, localizado no LAD. Os códigos foram executados 10x à fim de se obter um cálculo de média, utilizando 2,4,6 e 8 *threads*

2. Resultados

A Figura 1 apresenta os resultados de *speed-up* e eficiência do algoritmo de multiplicação de matrizes. O algoritmo apresenta um *speed-up* próximo ao ideal e a utilização dos recursos ficaram acima de 70%, aceitável para esse tipo de paralelismo. Durante as execuções, foi utilizado o método de escalonamento dinâmico com a definição do *chuck_size*, pois foi o que apresentou os melhores resultados. O método para definir o tamanho do *chunk_size* é através de um cálculo de divisão do tamanho da matriz pela quantidade de núcleos. Porém, essa simples divisão acaba sendo um problema quando ela não se tornava exata, causando um tempo de processamento maior. Esse problema foi contornado com uma instrução de verificação que arredonda o tamanho do *chuck_size*, antes de repassa-lo para as funções do *parallel for*.

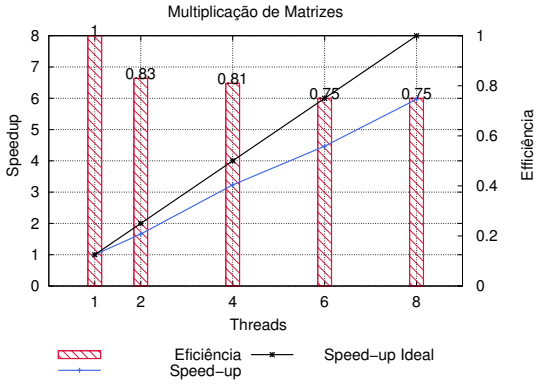

Figure 1. Desempenho do Código de Multiplicação de Matrizes.

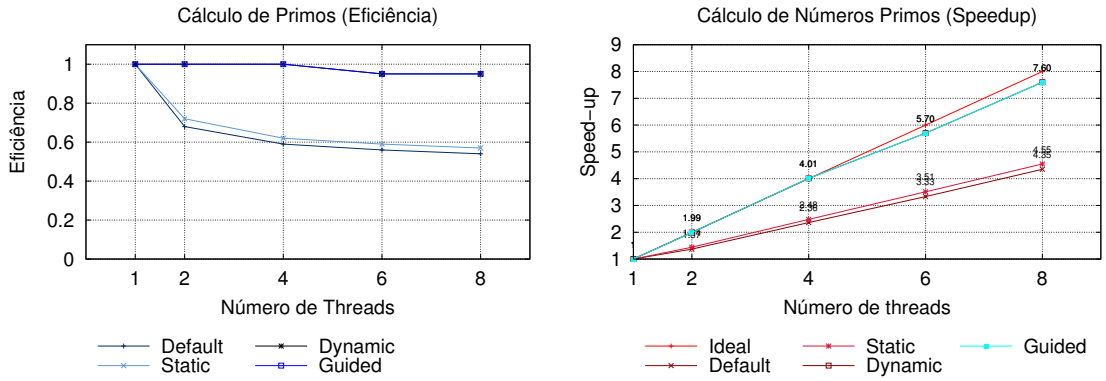

Figure 2. Desempenho dos Métodos de Escalonamento no OpenMP.

A Figura 2 apresenta os resultados dos tipos de escalonamento do OpenMP utilizando um algoritmo de cálculo de números primos. O algoritmo é executado intercalando os métodos de escalonamento através de chamadas de funções, e estas são seguidas pela padrão, a *static*, *dynamic* e *guided*.

Para a execução dos testes, o tamanho do *chuck_size* foi definido antes da inicialização das funções. Apenas o método *static* não recebeu esse parâmetro, sendo definido automaticamente durante a compilação do código. Dessa forma, pode-se perceber que o resultado do *static* foi próximo aos testes que não foi usado diretiva de escalonamento e o escalonamento *guided* nesta carga de trabalho utilizada teve o desempenho próximo ao *dynamic*.

3. Conclusões

Neste trabalho, foram apresentados alguns resultados da paralelização de algoritmos utilizando a biblioteca OpenMP. Todos os testes e implementações realizadas neste trabalho serviram para o estudos de algumas das principais diretivas da biblioteca de programação.

Algumas das dificuldades encontradas na realização do trabalho, foram as divisões dos trabalhos para cada *thread* do processador. Pois de acordo com a granularidade das tarefas, essa quantia poderia fazer o tempo de execução aumentar. Outra questão que poderia ser melhor explorado, é a definição do *chuck_size* no código de cálculo de números primos. Para realização dos testes, foi definido um valor fixo para esse parâmetro, porém, poderia trazer um resultado não satisfatório com um valor de entrada diferente.