UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
SISTEMAS DE INFORMAÇÃO

INE5645 - Programação Paralela e Distribuída

Profº Dr. Odorico Machado Mendizabal

Alan Vinicius Cezar Ensina (16206354)

A.7 OpenMP: Contador de palavras

A análise a seguir mostra a comparação de dois programas que foram desenvolvidos para contar quantas vezes certas palavras se repetem dentro de um texto. Sendo um código para o processamento totalmente sequencial e o outro sendo de processamento paralelo utilizando OpenMP.

Processamento sequencial

O primeiro código analisado foi o de processamento sequencial. O primeiro texto analisado possui 1222 caracteres, 250 palavras e 22 linhas.

Seu tempo de processamento foi de **0,000230s:**

```
------------------------------------------------------------
Tempo total de execução = 0.000230
------------------------------------------------------------
```

Ainda utilizando o mesmo código, foi mudado o texto de entrada para um outro texto com 647 caracteres, 129 palavras e 14 linhas, seu tempo de processamento foi **0,000148s:**

```
------------------------------------------------------------
Tempo total de execução = 0.000148
------------------------------------------------------------
```

Processamento paralelo utilizando OpenMP

A única mudança no código foi a inclusão da diretiva *#pragma omp parallel for* antes do laço de repetição que chama a função para checar a palavra e também foi setado como 4 o número de threads para que utilizasse uma thread para cada laço de repetição:

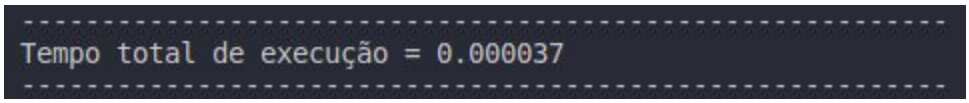
```
160   gettimeofday(&t1, NULL);
161
162   omp_set_num_threads(4);
163
164   #pragma omp parallel for
165   for(int i = 1 ; i <= 4 ; i++){
166     if(i == 1){
167       checaPalavra(palavra1, i);
168     }
169     if(i == 2){
170       checaPalavra(palavra2, i);
171     }
172     if(i == 3){
173       checaPalavra(palavra3, i);
174     }
175     if(i == 4){
176       checaPalavra(palavra4, i);
177     }
178   }
179   gettimeofday(&t2, NULL);
```

O primeiro texto utilizado como entrada é o mesmo do de processamento sequencial, onde possui 1222 caracteres, 250 palavras e 22 linhas.

Seu tempo de processamento foi de **0,000081s:**

```
------------------------------------------------------------
Tempo total de execução = 0.000081
------------------------------------------------------------
```

Ainda utilizando o mesmo código utilizando OpenMP, foi mudado o texto de entrada para o mesmo utilizado no sequencial, com 647 caracteres, 129 palavras e 14 linhas, seu tempo de processamento foi **0,000037s:**



Speed-Up

Para sabermos o speed-up dos dois códigos, é necessário utilizar a seguinte divisão:

$$Speedup(n) = \frac{tempo\_sequencial}{tempo\_paralelo(n)}$$

Os tempos coletados foram:

Texto 1 (sequencial): 0,000230s
Texto 1 (paralelo): 0,000081s
Texto 2 (sequencial): 0,000148s
Texto 2 (paralelo): 0,000037s

Speed-up (Texto 1): 0,000230/0,000081 = **2,839**
Speed-up (Texto 2): 0,000148/0,000037 = **4,00**

Considerações finais

Analisando as duas formas citadas acima, se fossemos optar pela opção de implementação mais fácil, logo escolheríamos a sequencial pois não teríamos que nos preocupar com gerenciamento de threads e nem de uma nova diretiva (no caso a OpenMP).

Caso o cenário exigisse um processamento paralelo e de fácil implementação, o OpenMP tem uma implementação muito mais fácil do que a implementação de POSIX threads.

A respeito do Speed-Up, podemos notar que no texto 1, o desempenho paralelo foi aproximadamente 3 vezes mais rápido que o sequencial. Já no texto 2, o desempenho paralelo foi 4 vezes mais rápido que o sequencial.

Link do repositório no Github

Repositório da tarefa no Github