**Processamento Paralelo com OpenMP - parte 1**

**OpenMP** é uma API multiplataforma que foi criada para suportar o desenvolvimento de algoritmos de processamento paralelo. Sua primeira versão é de 1997 para a linguagem Fortran. Um ano depois foi criada a versão para C e C++.

O modelo de programação utilizado na OpenMP é baseado em diretivas de compilação. Não há instruções nativas da linguagem (Fortran, C ou C++) inseridas no algoritmo para acionar as threads. O programador vai indicar ao compilador em que ponto devem ser instanciadas threads através de uma diretiva específica. Essas diretivas são denominadas pela palavra chave "**pragma**".

Salvo especificação em contrário, que determine a quantidade de threads para resolver determinado problema, o compilador irá preparar o instanciamento da quantidade igual à dos processadores existentes no computador.

Vamos ver o seguinte *Hello World!*:



Na linha 4 está a diretiva #**pragma** com os parâmetros **omp parallel**, onde *omp* vem de OpenMP; *parallel* indica que a partir deste ponto deve ser iniciado o processamento paralelo com cada thread fazendo exatamente a mesma coisa. Observe que o trecho a ser paralelizado é delimitado por chaves.

Utilizando o repl.it, não ocorre o paralelismo caso você for executar diretamente no botão *run*. É necessário ir à tela terminal e executar o g++.

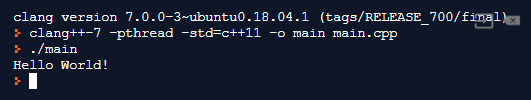


Figura 1 - Resultado da execução utilizando o botão "run".

O seu arquivo deve se chamar main.cpp, mas se quiser conferir pode dar o comando ls (ou ls -l) para listar os arquivos do diretório.

É necessário acrescentar o parâmetro de compilação **-fopenmp**.

Vamos chamar o executável de **exec**.

g++ -o exec main.cpp -fopenmp

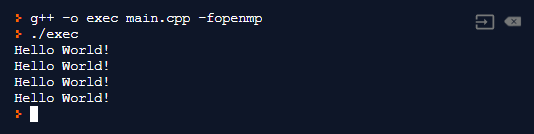


Figura 2 - Compilando com o parâmetro -fopenmp

Observe que no Linux, para execução de um programa que esteja no arquivo corrente, inicia-se o comando com "**./**".

O programa executou 4 vezes a mesma instrução. Para verificar quantos núcleos tem o computador que foi alocado no repl.it (ou em qualquer instalação Linux), digite o comando **lscpu**.

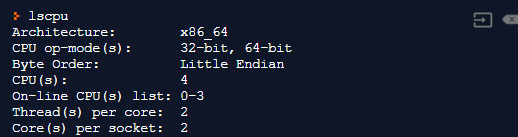


Figura 3 - Comando "lscpu"

A **Figura 3** mostra o resultado (parcial) do comando *lscpu*. Vê-se na linha 4 da saída que são apontadas 4 CPUs. Logo, nosso compilador preparou o programa para que fossem instanciadas 4 threads.

Caso se queira fixar um determinado número de threads, utiliza-se o complemento **num\_threads(N)**, onde N é o número desejado de threads.

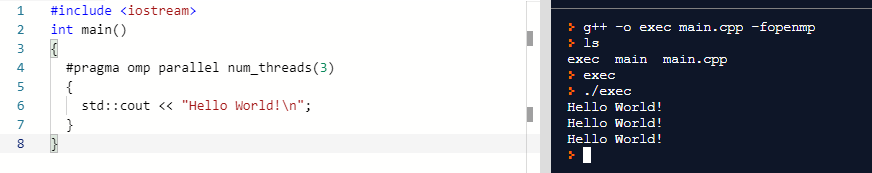


Figura 4 - Programa anterior com a especificação forçada em 3 threads.

Vamos analisar agora um exemplo com um bloco iterativo que desejamos paralelizar.

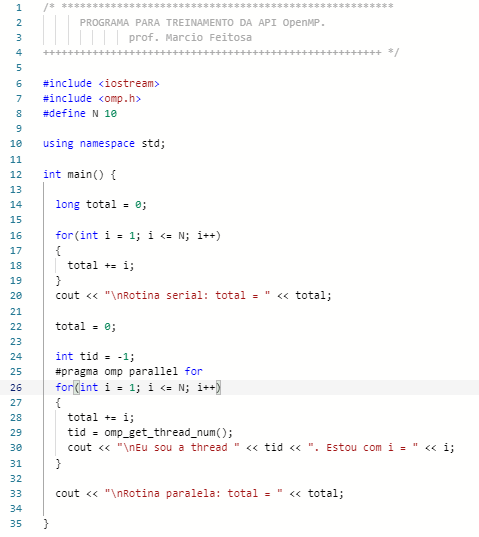


Figura 5 - Paralelismo no laço "for"

Observe que foi incluída a biblioteca "omp.h".

Neste algoritmo pretende-se fazer a somatória da sequência numérica de 1 até *N*. Para isso foi implementado um laço *for* e a somatória deverá ser armazenada na variável *total* (linha 14).

Primeiramente foi implementado um laço monoprocessado (linha 16) e na linha 20 é impresso o resultado da somatória.

Na linha 25 é reimplementado o mesmo laço, porém com a diretiva **parallel for**. O efeito desta diretiva é o seguinte: as threads vão dividir os números (i) para somar à variável *total*. Quando a thread completa o laço ela recebe o próximo valor i da sequência de todo o conjunto de threads, ou seja, se ela processou o valor 4, a próxima thread vai processar o valor 5, porque se trata da sequência do conjunto e não de cada thread individualmente.

O resultado de uma das execuções foi o seguinte:

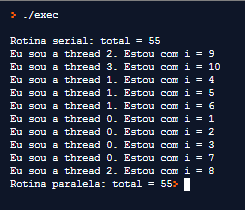


Figura 6 - Resultado de uma das execuções

O ID da thread é obtido com a instrução **omp\_get\_thread\_num()** (esta é a instrução que requer a biblioteca omp.h). Observe que o valor total, calculado pelo laço serial resultou em 55 e que no paralelo também. E também observe que as threads se revezaram e o valor de i não se repetiu.

Agora, faça a seguinte experiência para ver o resultado:

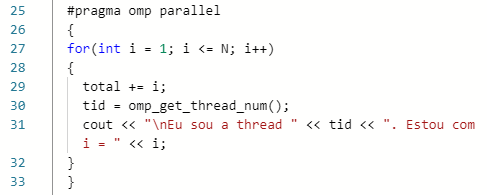


Figura 7

O que foi feito foi implementar o bloco paralelo na estrutura simples, conforme o exemplo anterior (observe as duas chaves adicionais para delimitar o bloco). Neste caso cada thread sairá executando o seu laço de forma independente. Quanto que você acha que vai resultar o total?

**Cláusula critical**

Uma observação a ser feita é que podem ocorrer eventuais truncamentos na saída (cout), pois não utilizamos nenhum tipo de bloqueio (mutex).

Na OpenMP existe o complemento **critical**. O que estiver envolvido pela diretiva #pragma omp critical só executado por uma thread de cada vez, não ocorrendo, neste trecho, nenhum tipo de paralelismo. Observe o código abaixo:

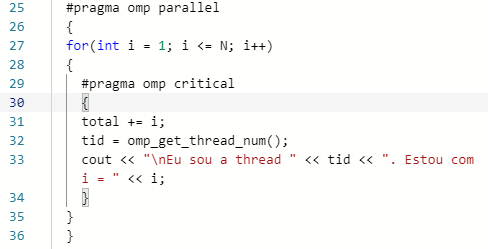


Figura 8

Aqui foi envolvido todo o conteúdo do laço *for* pela diretiva "critical". Dessa forma não ocorrerá nenhum truncamento nos "prints" de saída[[1]](#footnote-1).

**Cláusula section**

Para encerrarmos esta parte 1, vamos ver a cláusula **section**. Com esta cláusula podemos paralelizar partes do código, que chamaremos de tarefas, que sejam independentes, ou seja, o início de uma não depende de resultados obtidos por outra.

Vamos analisar o seguinte código:

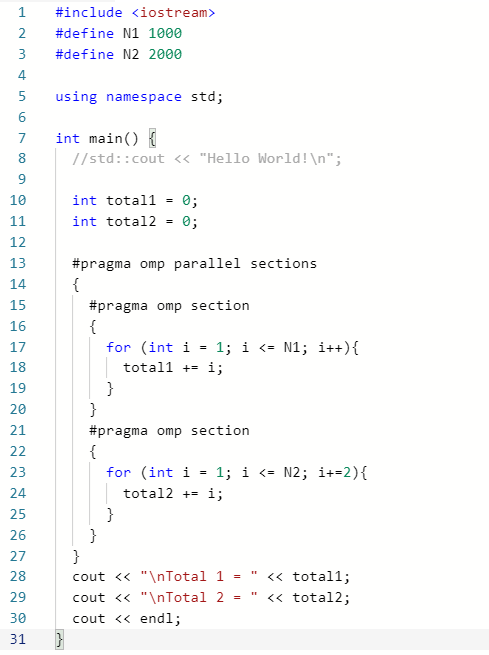


Figura 9

Na linha 13 é declarado o início do conjunto (ou de um dos conjuntos) de blocos section. No caso temos 2 sections neste conjunto. Ambas executam um laço for, mas o primeiro faz a somatória de 1 a N1 e o segundo de 1 a N2, porém com passo 2.

Será instanciada uma única thread para executar o primeiro bloco e uma outra para executar o segundo bloco.

Convém ressaltar que no bloco **#pragma omp parallel sections** não pode haver nenhuma instrução que não esteja dentro de um bloco **#pragma omp section**.

Como parte 1 já é suficiente. A seguir vamos ver que exercícios deveremos fazer.

**Para utilizar as técnicas de paralelismo da OpenMP.**

**Exercício 1:**

No algoritmo, crie 4 vetores de números inteiros, "popule-os" com números aleatórios e os ordene com o método bubble-sort (o menos eficiente). Insira as instruções para capturar o tempo inicial e o tempo final do processo. Imprima o tempo decorrido. Você deve ajustar o tamanho dos vetores para que o tempo de processamento seja observável e também não muito extenso.

Salve este código. Como parte II insira as diretivas para paralelizar a ordenação de cada um dos vetores. Imprima o tempo de processamento.

Você deve entregar estes dois códigos.

**Exercício 2:**

No algoritmo, crie uma matriz quadrada (N X N), "popule-a" com números aleatórios e faça um duplo *for* para contar os elementos maiores que determinado valor (escolha você esses parâmetros). Insira as instruções para capturar o tempo inicial e o tempo final do processo. Imprima o tempo decorrido.

Salve este código. Como parte II insira as diretivas de paralelização do laço *for*.

100011101010010011111

1. Obviamente que neste exemplo o paralelismo foi comprometido na sua quase totalidade, pois a quase totalidade do bloco "parallel" está sendo executada sob a cláusula "critical". [↑](#footnote-ref-1)