**Processamento Paralelo com OpenMP - parte 2**

**Cláusula shared/private**

Por padrão (caso não haja declaração explícita) as variáveis declaradas fora das threads são compartilhadas (shared) e as internas são privadas (private). As variáveis *shared* tem uma cópia única que todas as threads acessam e as variáveis *private* são replicadas, ou seja, cada thread tem a sua[[1]](#footnote-1) e outras não podem modificar.

No entanto, o status *default* pode ser modificado de forma explícita pelas cláusulas shared e private.

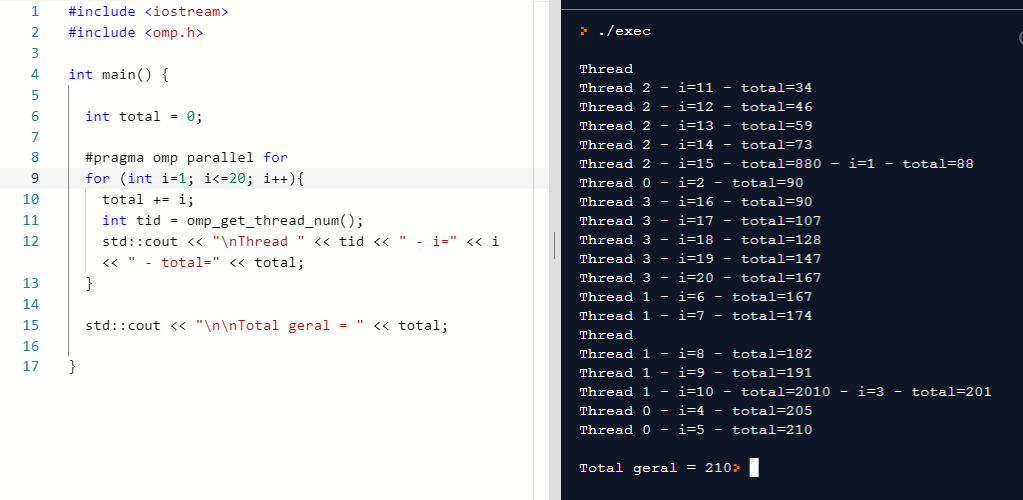


Figura 1

Na temos um programa sem especificação shared/private (já vimos na aula passada). Por default, a variável "total" é *shared* (declarada fora da thread) e a somatória de todas as threads nesta variável total (linha 10) resulta um valor único global.

Vamos ver agora o que acontece quando se declara variável total como *private*:

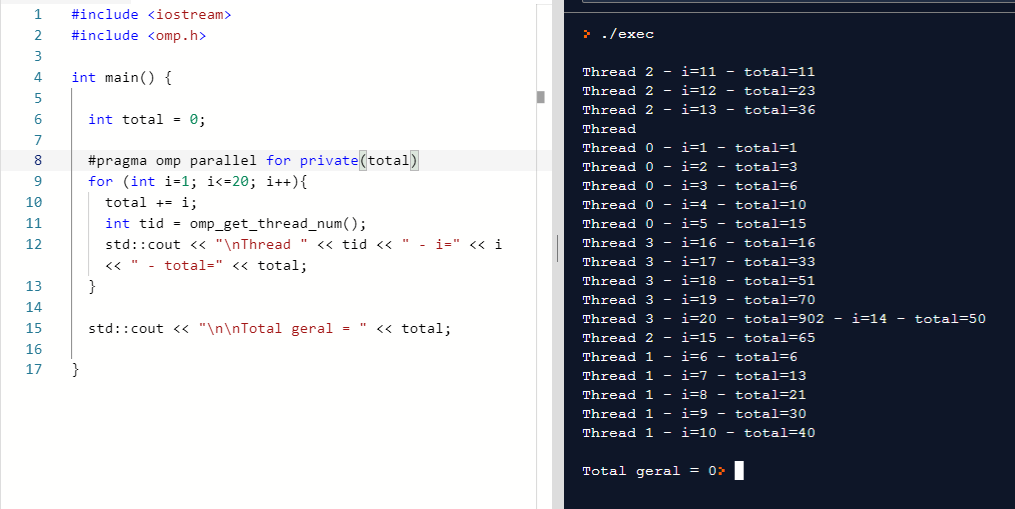


Figura 2

Na linha 8 foi declarada a variável "total" como sendo private. Ocorreu que cada thread criou uma instância própria desta variável e percebe-se, na saída à direita, que a variável "total" original não foi afetada. Na saída vê-se que o último registro da thread 0 resultou 15, thread 1 = 40, thread 2 = 65 e thread 3 = 90.

**Execução paralela condicional**

Pode-se estabelecer que determinado trecho do algoritmo execute em paralelo ou não de acordo com determinada condição.

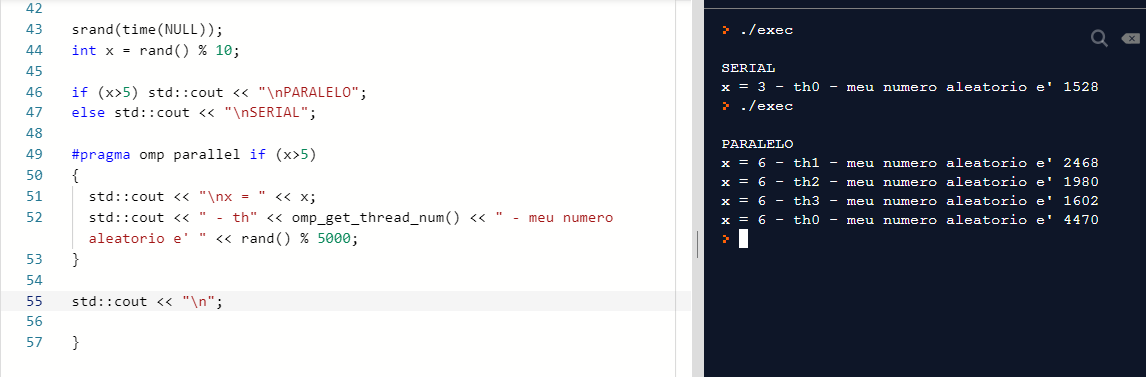


Figura 3

Neste exemplo, linha 49[[2]](#footnote-2), indica que o paralelismo só será disparado se o valor de x for maior que 5. Na saída tem-se 2 execução consecutivas, a primeira gerou o número 3. Não atende a condição if, logo será executado mono-thread (thread 0). A segunda gerou o número 6 que atende a condição if, logo será executado em multi-thread.

**A cláusula reduction**

Indica que determinada operação deverá ser executada com as variáveis das threads.

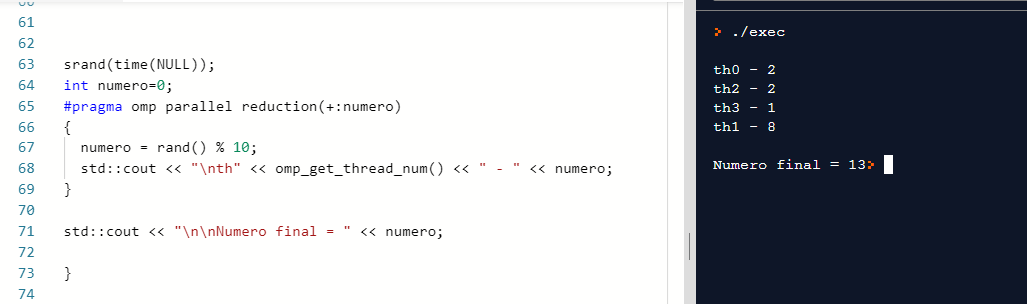


Figura 4

No código da , linha 65, temos a cláusula **reduction(op:vars)**, onde **op** é o tipo de operação e **vars** são as variáveis a serem reduzidas pela operação. No caso do exemplo, op=+, ou seja, as variáveis serão somadas, e as variáveis a serem somadas, no caso só foi listada uma, numero.

Na saída vê-se o resultado. Cada thread gerou um número aleatório e no final é impressa a variável com o valor da soma de todas.

A cláusula reduction torna as variáveis private e deposita o resultado na shared.

**A cláusula schedule**

Nos laços for o próximo valor "i" é dado à próxima thread que solicitar. Pode-se, no entanto, especificar a quantidade de "is" (em sequência) a serem delegadas à próxima thread que solicitar.

Há 3 tipos de schedule: static, dynamic e guided.

No modo static a distribuição dos "is" segue a filosofia das "cartas marcadas", ou seja, o primeiro conjunto é para a thread 0, o segundo é para a thread 1, etc. independentemente da sequência de execução.

No modo dynamic, a próxima query que pedir recebe o próximo conjunto de "is", não há a "carta marcada".

O modo guided opera como o modo dynamic, porém o tamanho da porção de "is" é controlado pelo ambiente, sendo que a quantidade especificada será sempre o mínimo.

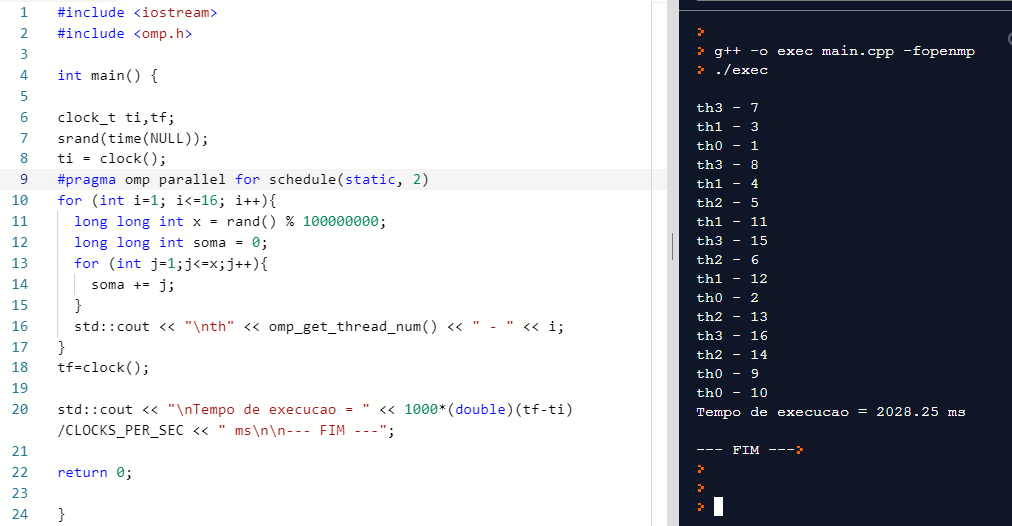


Figura 5

No código da , linha 9, está especificado que o schedule é static e o conjunto de "is" é 2. Vê-se na saída que:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i ->** | 1,2 | 3,4 | 5,6 | 7,8 | 9,10 | 11,12 | 13,14 | 15,16 |
| **thread** | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |

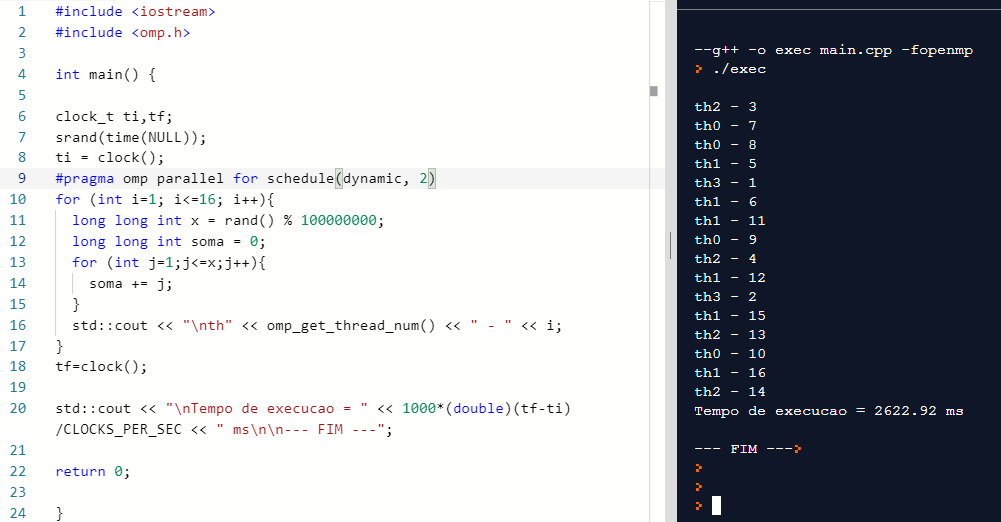


Figura 6

No código da , linha 9, está especificado que o schedule é dynamic e o conjunto de "is" é 2. Vê-se na saída que:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i ->** | 1,2 | 3,4 | 5,6 | 7,8 | 9,10 | 11,12 | 13,14 | 15,16 |
| **thread** | 3 | 2 | 1 | 0 | 0 | 1 | 2 | 1 |

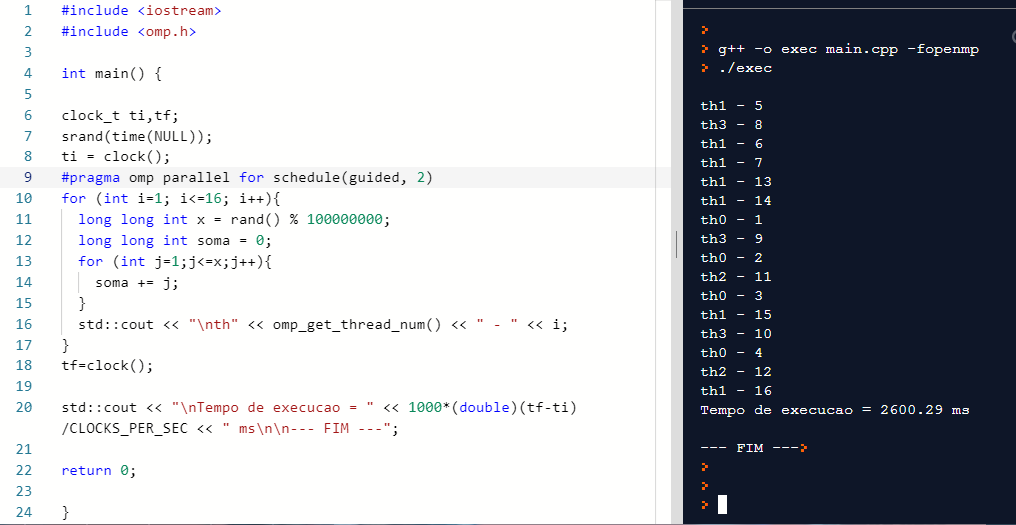


Figura 7

No código da , linha 9, está especificado que o schedule é guided e o conjunto mínimo de "is" é 2. Vê-se na saída que:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **i ->** | 1,2,3,4 | 5,6,7 | 8,9,10 | 11,12 | 13,14,15,16 |
| **thread** | 0 | 1 | 3 | 2 | 1 |

**A cláusula single**

Esta cláusula delimita um setor da thread que deve ser executado por apenas uma das threads do time.

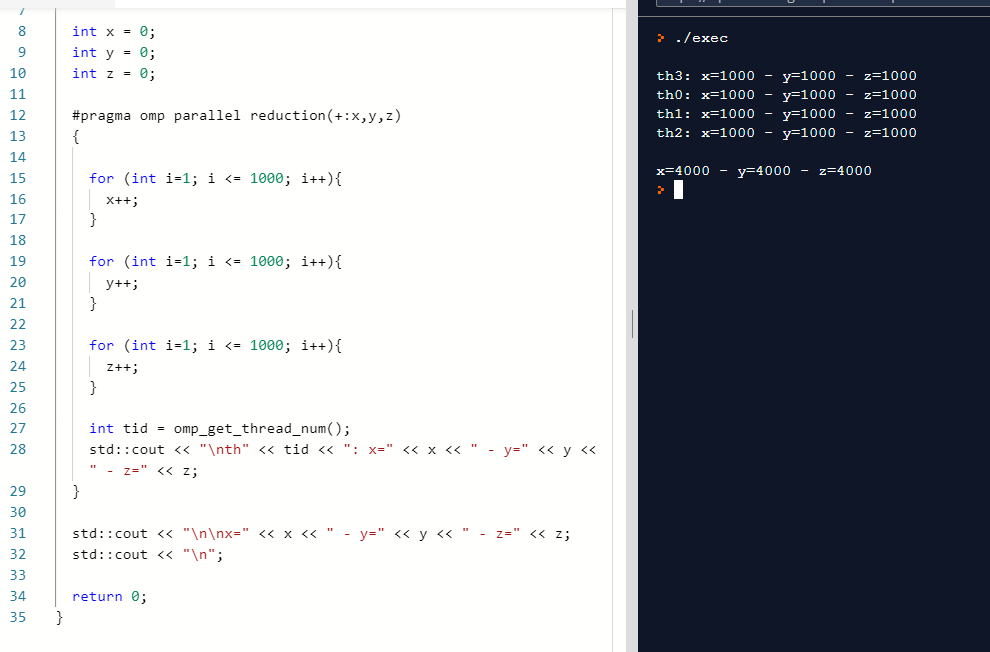


Figura 8

No código da tem-se um bloco paralelo que executa 3 laços for seguidos incrementando uma variável de uma unidade. Na saída à direita pode-se ver o resultado.

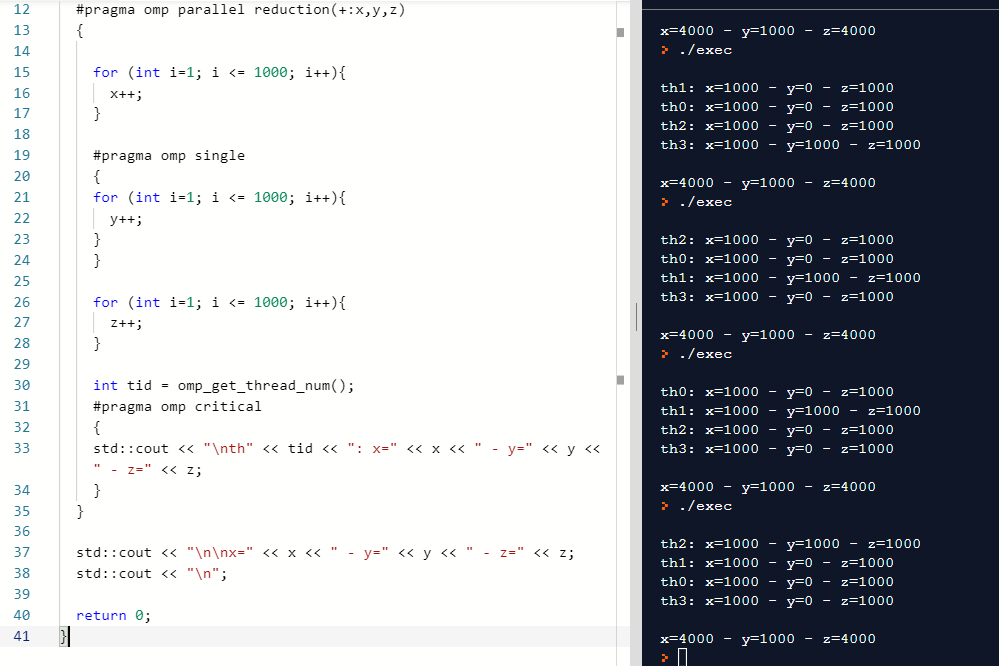


Figura 9

Já neste código da **Figura 9** foi inserida a diretiva single no bloco que incrementa a variável y (linha 19). Observa-se na saída que apenas uma das threads do grupo que faz esta operação, nas demais a variável permanece com valor zero. A thread é escolhida aleatoriamente.

**A cláusula master**

Tem o mesmo resultado da cláusula single, porém sempre com a mesma thread. A thread 0. Faça o teste.

Exercício:

1. O algoritmo deve instanciar 3 matrizes N X N (as 3 com as mesmas dimensões).
2. "Popule" essas 3 matrizes com números aleatórios.
3. Utilizando um processo sequencial, some todos os elementos de cada matriz e, em seguida, some os 3 resultados. Armazene esta soma global.
4. Em seguida, utilizando as técnicas de paralelização estudadas, execute este cálculo pelos seguintes passos:
   1. Em cada matriz, some os elementos de cada linha e obtenha 3 vetores de N posições cada um com essas somas.
   2. Ordene os vetores (utilize o algoritmo que preferir).
   3. Some as posições correspondentes dos vetores e obtenha um único vetor final.
   4. Some os elementos do vetor final.

Muito bem, se o seu algoritmo estiver correto esse resultado final deve ser o mesmo do primeiro obtido pelo processo sequencial.

*100101011110010100100101*

1. Pode-se fazer uma analogia com as classes (POO) que possuem atributos estáticos, onde esses atributos são comuns a todas as instâncias da classe, e os atributos não estáticos são replicados individualmente. [↑](#footnote-ref-1)
2. O programa inicia na linha 43. O código acima é de teste e foi anulado. [↑](#footnote-ref-2)