Diferença entre as diretivas atomic e critical

Sintaxe

```
#pragma omp atomic
   expression
```

Um nome (opcional) para identificar o código critical. Observe que esse nome deve estar entre parênteses.

```
#pragma omp critical [(name)]
{
   code_block
}
```

Exemplos

```
#pragma omp atomic
g_qCount++;


#pragma omp critical
g_qCount++;
```

O efeito em g_qCount é o mesmo, mas a forma como funcionam é diferente.

A diretiva “critical” do OpenMP é completamente genérica - pode cercar qualquer bloco arbitrário de código. Ao passo que em uma diretiva “atomic” existe uma única expressão.

Além disso, no OpenMP numa diretiva “critical” sem nome, as seções críticas são consideradas idênticas, de modo que se uma thread estiver em uma seção crítica (sem nome) como acima, nenhuma outra thread pode executar essa seção crítica.

Você pode contornar isso usando seções críticas nomeadas (veja sintaxe).

Uma operação ‘’atomic’’ tem uma sobrecarga muito menor. A diferença entre as duas é que a "atomic" tenta utilizar recursos do hardware para ser mais eficiente. Por exemplo, uma operação de incremento atômico.

As construções “critical” e “atomic” servem para dizer que o que estiver dentro do escopo dessas diretivas, deve ser executado somente por uma *thread*, num determinado instante de tempo.

```
#pragma omp parallel for
for(int i = 0; i < 1048576; ++i)
{
  int partial_result = f(x[i], y[i], z[i]);

  #pragma omp atomic
  {
    sum += partial_result;
  }
}
```

Outro exemplo:

```
#include <stdio.h>
#include <omp.h>

#define MAX 10

int main() {
   int count = 0;
   #pragma omp parallel num_threads(MAX)
   {
      #pragma omp atomic
      count++;
   }
   printf_s("Number of threads: %d\n", count);
}
```

Output

```
Number of threads: 10
```

Diretiva single

Além dessas, existe também a diretiva "single". Esta especifica que o bloco de código deve ser executado por apenas uma das threads.

Sintaxe

```
#pragma omp single [clauses]
{
    code_block
}
```

A diretiva “single” suporta as seguintes cláusulas OpenMP:
copyprivate, firstprivate, nowait, private

Exemplo

```
#include <stdio.h>
#include <omp.h>

int main() {
   #pragma omp parallel num_threads(2)
   {
      #pragma omp single
      // Only a single thread can read the input.
      printf_s("read input\n");

      // Multiple threads in the team compute the results.
      printf_s("compute results\n");

      #pragma omp single
      // Only a single thread can write the output.
      printf_s("write output\n");
    }
}
```

Output

```
read input
compute results
compute results
write output
```

Diretiva master

A diretiva "master" é parecida: nela é garantido que a thread que vai executar aquele bloco é a *master thread*. Isso pode ser útil em alguns casos. Por exemplo, código para atualizar uma interface gráfica normalmente só pode ser executado em uma thread específica, que é a principal. Se você quiser atualizar a tela, por qualquer motivo, de dentro de uma seção paralela, tem que garantir que essa atualização será feita pela master thread. Por exemplo:

```
#pragma omp parallel
{
        a();
        b();
        c();
        #pragma omp barrier
        #pragma omp master
        {
                send_results_to_screen();
        }
        d();
        e();
        f();
}
```

Diretiva section

https://pt.wikibooks.org/wiki/Programação_Paralela_em_Arquiteturas_MultiCore/Programação_em_OpenMP

Sintaxe

```
#pragma omp [parallel] sections [clauses]
{
   #pragma omp section
   {
      code_block
   }
}
```

"omp sections" faz a construção de divisão de trabalho. Ela define blocos independentes, que podem ser distribuídos entre as *threads*. Cada *section* vai ser executada apenas por uma *thread*. Trata-se de uma forma simples de paralelizar tarefas distintas que não tem (ou tem pouca) dependência de dados entre si.

Um exemplo simples de uso seria a inicialização de dois arranjos com conteúdos distintos e que podem ser inicializado de maneira independente um do outro.

Exemplo:

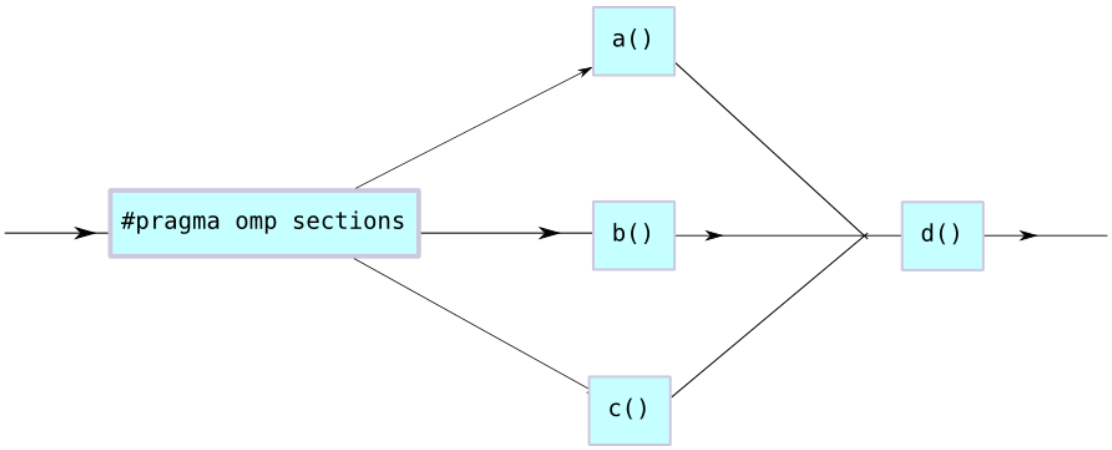
```
#pragma omp parallel sections
{
        #pragma omp section
        {
                a();
        }

        #pragma omp section
        {
                b();
        }

        #pragma omp section
        {
                c();
        }
}

d();
```

Nesse exemplo, as funções a, b e c seriam executadas em paralelo, enquanto a função d só seria chamada após todas as seções paralelas terem sido finalizadas.



]

Cláusula nowait

Sempre que uma thread em uma construção de compartilhamento termina seu trabalho mais rápido do que outras threads, essa thread aguardará até que todas terminem o trabalho respectivo. Todos as threads serão sincronizadas no final da construção de compartilhamento de trabalho.

Para situações em que não precisamos ou queremos sincronizar, OpenMP fornece a cláusula “nowait”

Sintaxe

```
nowait
```

Exemplo

```
#include <stdio.h>

#define SIZE 5

void test(int *a, int *b, int *c, int size)
{
    int i;
    #pragma omp parallel
    {
        #pragma omp for nowait
        for (i = 0; i < size; i++)
            b[i] = a[i] * a[i];

        #pragma omp for nowait
        for (i = 0; i < size; i++)
            c[i] = a[i]/2;
    }
}

int main( )
{
    int a[SIZE], b[SIZE], c[SIZE];
    int i;

    for (i=0; i<SIZE; i++)
        a[i] = i;


    test(a,b,c, SIZE);

    for (i=0; i<SIZE; i++)
        printf_s("%d, %d, %d\n", a[i], b[i], c[i]);
}
```

Output

```
0, 0, 0
1, 1, 0
2, 4, 1
3, 9, 1
4, 16, 2
```