Exemplo Cláusula Schedule

https://msdn.microsoft.com/pt-br/library/x5aw0hdf.aspx

Sintaxe

```
schedule(type[,size])
```

Parâmetros

`type`
O tipo de agendamento:

- `dynamic`
- `guided`
- `static`

`size` (opcional)
Especifica o tamanho de iterações. `Size` deve ser um inteiro.

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

#define NUM_THREADS 4
#define STATIC_CHUNK 5
#define DYNAMIC_CHUNK 5
#define NUM_LOOPS 20
#define SLEEP_EVERY_N 3

int main( )
{
    int nStatic1[NUM_LOOPS],
        nStaticN[NUM_LOOPS];
    int nDynamic1[NUM_LOOPS],
        nDynamicN[NUM_LOOPS];
    int nGuided[NUM_LOOPS];

    omp_set_num_threads(NUM_THREADS);

    #pragma omp parallel
    {
        #pragma omp for schedule(static, 1)
        // O “loop” será dividido, no tempo de compilação,  em pedaços de tamanho
        //  “chunk=1” e distribuído estaticamente para cada “thread”. Se o parâmetro
        //  “chunk” não for definido, o compilador irá dividir o “loop”  em partes iguais,
        //  para cada “thread”.
        for (int i = 0 ; i < NUM_LOOPS ; ++i)
        {
            if ((i % SLEEP_EVERY_N) == 0)
                Sleep(0);
            nStatic1[i] = omp get thread num( );
        }

        // O “loop de 20 iterações” será dividido, no tempo de compilação, em agrupamentos
        //  de tamanho “STATIC_CHUNK = 5 iterações” e,  cada “thread” (0, 1, 2, 3) é
        //  alocada estaticamente a um  agrupamento. Ou seja, cada thread (0, 1, 2, 3)
        //  executa em um agrupamento de 5 iterações.
        #pragma omp for schedule(static, STATIC_CHUNK)
        for (int i = 0 ; i < NUM_LOOPS ; ++i)
        {
            if ((i % SLEEP_EVERY_N) == 0)
                Sleep(0);
            nStaticN[i] = omp_get_thread_num( );
        }

        // Aqui, algumas das iterações são atribuídas, a um número
        // menor de threads. Após o término da iteração atribuída a um thread em particular,
        //  ele retorna para buscar uma das iterações restantes. O parâmetro "chunk=1"
        //  define o número de iterações sequenciais que são atribuídas a um thread por vez.
```

```
        #pragma omp for schedule(dynamic, 1)
        for (int i = 0 ; i < NUM LOOPS ; ++i)
        {
            if ((i % SLEEP_EVERY_N) == 0)
                Sleep(0);
            nDynamic1[i] = omp_get_thread_num( );
        }

        // Aqui, algumas das iterações são atribuídas a um número menor de threads. Após o
        // término da iteração atribuída a um thread em particular, ele retorna para buscar
        // uma das iterações restantes. O parâmetro "chunk" define o número de iterações
        // sequenciais que são atribuídas a um thread por vez.

        #pragma omp for schedule(dynamic, DYNAMIC CHUNK)
        for (int i = 0 ; i < NUM_LOOPS ; ++i)
        {
            if ((i % SLEEP EVERY N) == 0)
                Sleep(0);
            nDynamicN[i] = omp get thread num( );
        }

        // Um grande "chunk" de iterações sequenciadas são atribuídos a cada thread
        // dinamicamente (como acima). O tamanho do "chunk" diminui exponencialmente
        // com cada atribuição sucessiva até um tamanho mínimo especificado no parâmetro
        // chunk.
        #pragma omp for schedule(guided)
        for (int i = 0 ; i < NUM_LOOPS ; ++i)
        {
            if ((i % SLEEP_EVERY_N) == 0)
                Sleep(0);
            nGuided[i] = omp get thread num( );
        }
    }

    printf s("-----------------------------------------------\n");
    printf s("| static | static | dynamic | dynamic | guided |\n");
    printf s("|    1   |    %d   |    1    |    %d    |        |\n",
             STATIC CHUNK, DYNAMIC CHUNK);
    printf_s("-----------------------------------------------\n");

    for (int i=0; i<NUM LOOPS; ++i)
    {
        printf s("|    %d   |    %d   |    %d    |    %d    |"
                 "    %d   |\n",
                 nStatic1[i], nStaticN[i],
                 nDynamic1[i], nDynamicN[i], nGuided[i]);
    }

    printf s("-----------------------------------------------\n");
}
```

Output (Examine, por enquanto, os dois primeiros casos)

```
-----------------------------------------------
| static | static | dynamic | dynamic | guided |
|    1   |    5   |    1    |    5    |        |
-----------------------------------------------
|    0   |    0   |    0    |    2    |    1   |
|    1   |    0   |    3    |    2    |    1   |
|    2   |    0   |    3    |    2    |    1   |
|    3   |    0   |    3    |    2    |    1   |
|    0   |    0   |    2    |    2    |    1   |
|    1   |    1   |    2    |    3    |    3   |
|    2   |    1   |    2    |    3    |    3   |
|    3   |    1   |    0    |    3    |    3   |
|    0   |    1   |    0    |    3    |    3   |
|    1   |    1   |    0    |    3    |    2   |
|    2   |    2   |    1    |    0    |    2   |
|    3   |    2   |    1    |    0    |    2   |
```

```
|    0   |    2   |    1    |    0    |    3   |
|    1   |    2   |    2    |    0    |    3   |
|    2   |    2   |    2    |    0    |    0   |
|    3   |    3   |    2    |    1    |    0   |
|    0   |    3   |    3    |    1    |    1   |
|    1   |    3   |    3    |    1    |    1   |
|    2   |    3   |    3    |    1    |    1   |
|    3   |    3   |    0    |    1    |    3   |
-------------------------------------------------
```