Ciclo **for** con OpenMP

Carlos E. Alvarez[1].

[1]Dep. de Matemáticas aplicadas y Ciencias de la Computación, Universidad del Rosario

2020-I

Ciclo for

El constructo `#pragma omp for` reparte la ejecución de las iteraciones de un ciclo for entre los diferentes hilos.

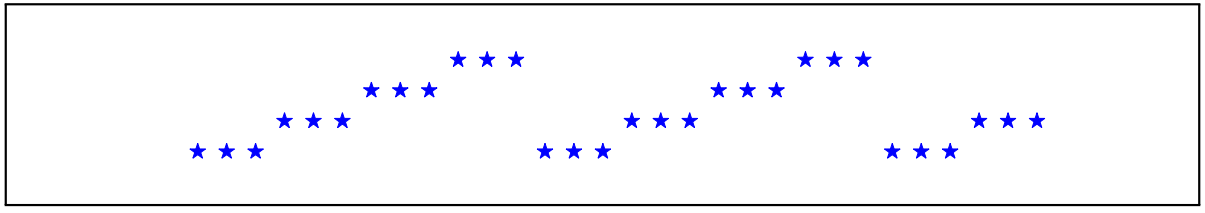
```
1       #pragma omp parallel
2       {
3         #pragma omp for
4         for(int i = 0; i < N; ++i){
5           .
6           .
7           .
8         }
9       }
```

¿De qué manera se reparte el trabajo? Clausula `schedule`:

- `schedule(static, chunk_size)`
- `schedule(dynamic, chunk_size)`
- `schedule(guided, min_chunk_size)`
- `schedule(auto)`
- `schedule(runtime)`

Agendamiento estático

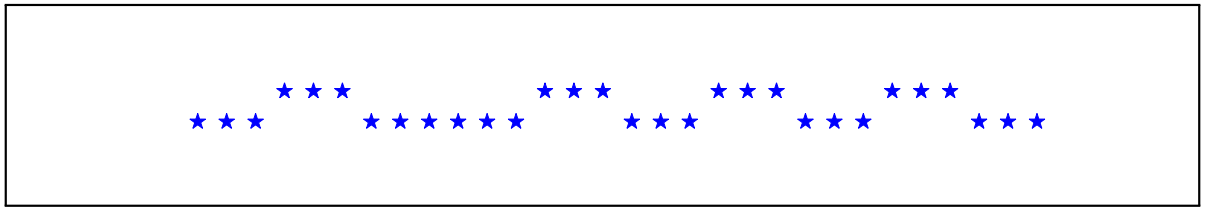
La cláusula `schedule(static, chunk_size)` hace que cada uno de los hilos procese secciones de tamaño `chunk_size` en orden ascendente.



Ver ejemplo proporcionado.

Agendamiento dinámico

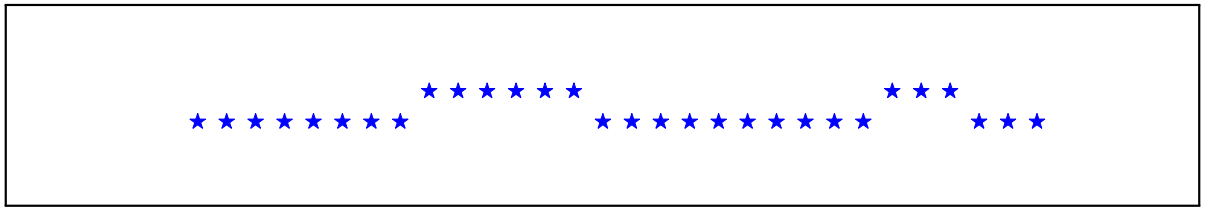
La cláusula `schedule(dynamic, chunk_size)` hace que los hilos procesen secciones de tamaño `chunk_size`. El orden en el que lo hacen es determinado durante la ejecución por el algoritmo agendador (scheduler). Útil cuando las iteraciones tienen costos computacionales no homogeneos.



Ver ejemplo proporcionado.

Agendamiento guiado

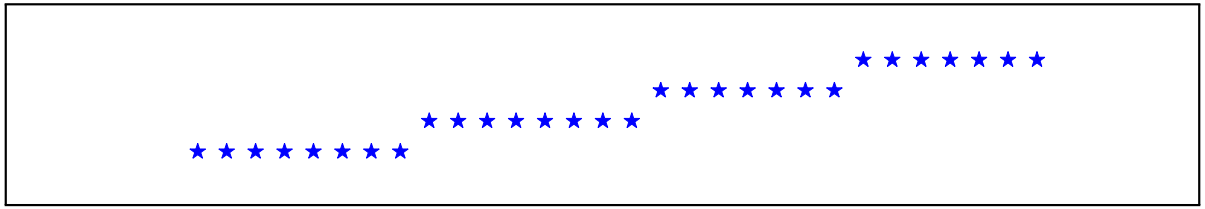
La cláusula `schedule(guided, min_chunk_size)` hace que el agendador (scheduler) decida el tamaño del chunk y el orden en el que se ejecutan los hilos. El tamaño mínimo de un chunk está dado por `min_chunk_size`. Útil cuando las iteraciones tienen costos computacionales no homogeneos.



Ver ejemplo proporcionado.

Agendamiento automático

La cláusula `schedule(auto)` hace que el orden y el tamaño de las secciones procesadas por los hilos sea elegida automáticamente por el agendador (scheduler).



Ver ejemplo proporcionado.

Agendamiento en tiempo de ejecución

La cláusula `schedule(runtime)` permite obtener el tipo de agendamiento y tamaño del `chunk` durante la ejecución, leyéndolo de la variable de ambiente `OMP_SCHEDULE`.

Asignando `OMP_SCHEDULE` (bash)

```
1       export OMP_SCHEDULE="dynamic, 3"
```

Ver ejemplo proporcionado.

Ciclos anidados

Para ciclos anidados rectangulares $(m \times n)$ es posible usar la cláusula `collapse()`, que organiza las iteraciones de forma lineal.

```
1     #pragma omp for collapse(2)
2     for(int i = 0; i < M; ++i){
3       for(int j = 0; j < N; ++j){
4         .
5         .
6         .
7       }
8     }
```

Útil si `M` es el número de unidades de ejecución.

Ej: La norma de Frobenius de una matriz consiste en la raiz cuadrada de la suma de todos sus elementos al cuadrado:

$$||A||_F = \sqrt{\sum_i^M \sum_j^N a_{ij}^2} \tag{1}$$

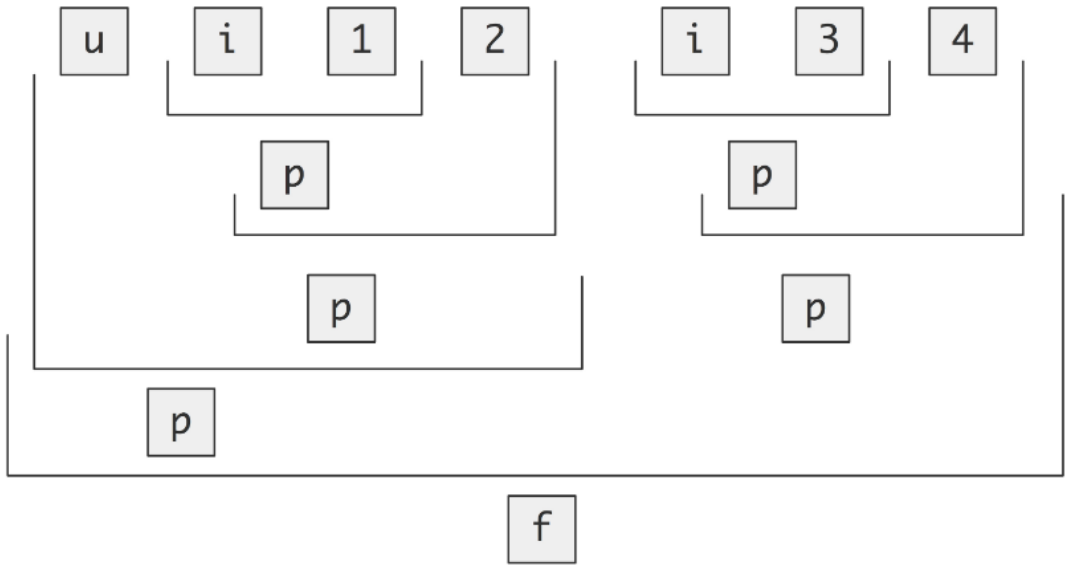
- Escriba una función serial `frobenius(int* A,int m,int n)` que reciva un arreglo dinámico que represente una matriz $m \times n$, y retorne su norma de Frobenius
- Escriba una versión paralela en donde se use la clausula `collapse` para paralelizar el ciclo doble
- Implemente estas funciónes en dos programas (serial y paralelo). Compruebe correctitud y compare tiempo de ejecución

Reduction

Otro método para modificar una variable compartida sin bloqueo es `reduction`. Nos permite realizar las siguentes operaciones: +,-,*,/,max,min.

```
1     #pragma omp for reduction(op: var)
2     for(int i = 0; i < N; ++i){
3       var op= ...
4     }
```

Ejemplo de algoritmo de reducción:



Cada bracket representa una operación, `u` representa la inicialización del usuario, `i` la inicialización del algoritmo, `p` un resultado parcial y `f` el resultado final.

Ej: Esciba una implementación del programa que calcula la norma de Frobenius usando `reduce`.