Programación en Paralelo

Por: Jesús Manuel Mager Hois

Asesor: Carlos Pineda Muñoz

¿Qué es la programación en paralelo?

La programación paralela es una forma de computación donde varios cálculos son efectuados de manera simultánea, bajo el principio de que cada problema puede ser separado en partes para su tratamiento. Este tiempo de programación es válido para equipos que tienen procesadores con varios núcleos (multicore) y computadoras con varios procesadores de memoria compartida.

¿Por qué usar la programación en paralelo?

Las nuevas computadoras, e incluso los dispositivos móviles han sido equipados con procesadores multicore, o incluso con varios procesadores. **La programación en paralelo explota al máximo el rendimiento de estas caracterísitcas de hardware.**

Equipos multicore

Un equipo multicore es **un procesador en un único componente con dos o más unidades centrales de procesamiento** llamados cores o núcleos, que son procesadores que ejecutan instrucciones. Estas instrucciones son instrucciones típicas de cada procesador, como puede ver add, mov, etc... que pueden ser ejecutadas simultáneamente por sus diferentes núcleos.

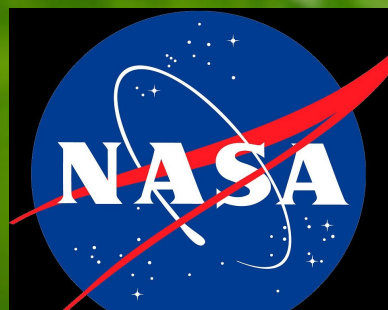
Memoria Compartida

Las computadoras paralelas de memoria compartida son aquellas donde los procesadores individuales comparten la entrada y salida de la memoria de tal manera que cada uno de ellos puede acceder a la memoria en cualquier locación con la misma velocidad, esto significa que tienen acceso uniforme a memoria.

OpenMP

- OpenMP es API multiplataforma para multiprocesos de memoria compartida. Está escrito para usarse de manera nativa desde C, C++ y Fortran. Es Open Sorce.
- Plataformas en que funciona: Windows, Mac OS X y la familia de sistemas operativos Unix, entre muchos otros.

Mesa de revisión de arquitectura de OpenMP



Un pequeño ejemplo en OpenMP

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

int main (int argc, char *argv[])
{
    int n_hilos, id_hilo;
    omp_set_num_threads(4);
    #pragma omp parallel private(id_hilo, n_hilos)
    {
        id_hilo = omp_get_thread_num();
        printf("Este es el hilo con identidad: %d\n", id_hilo);

        if (id_hilo == 0)
        {
            n_hilos = omp_get_num_threads();
            printf("Cantidad total de hilos activos: %d\n", n_hilos);
        }
    }
}
```

¿Cuáles son las directivas más comúnes?

- For
- Section
- Single
- Barrier
- Critical
- Atomic

A continuación veremos cada una de estas directivas.

Directiva for

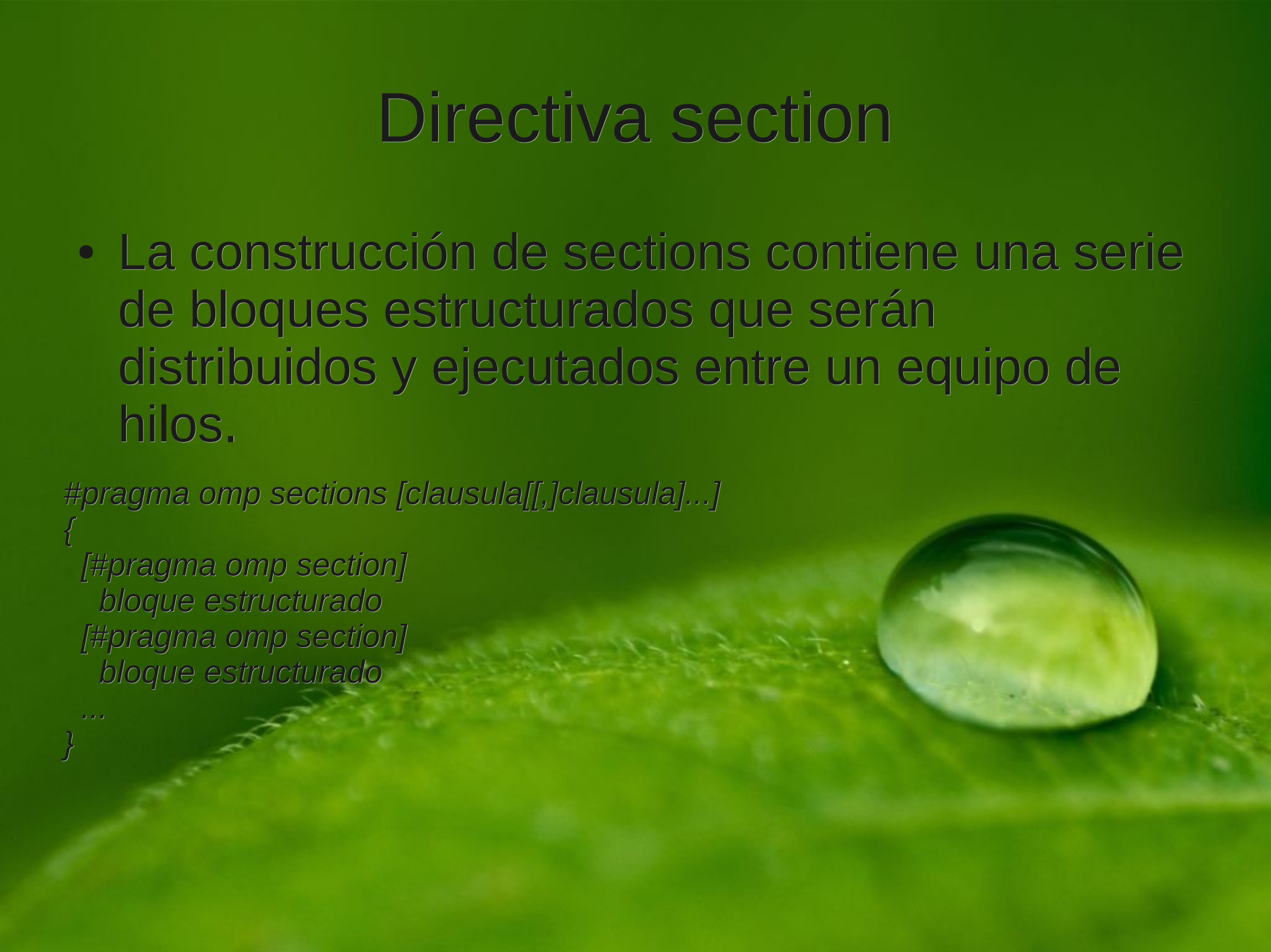
- La construcción for especifica que las iteraciones de los ciclos serán distribuidas y ejecutadas entre el equipo de hilos.

#pragma omp for [clausula[[,]clausula]...]

Directiva section

- La construcción de sections contiene una serie de bloques estructurados que serán distribuidos y ejecutados entre un equipo de hilos.

```
#pragma omp sections [clausula[[,]clausula]...]
{
  [#pragma omp section]
    bloque estructurado
  [#pragma omp section]
    bloque estructurado
  ...
}
```



Directiva single

- La construcción *single* especifica que el bloque estructurado asociado es ejecutado únicamente por uno de los hilos que componen el equipo de hilos(no necesariamente el hilo maestro), en el contexto de una tarea implícita.

#pragma omp single [clausula[[,]clausula]...]

Directiva barrier

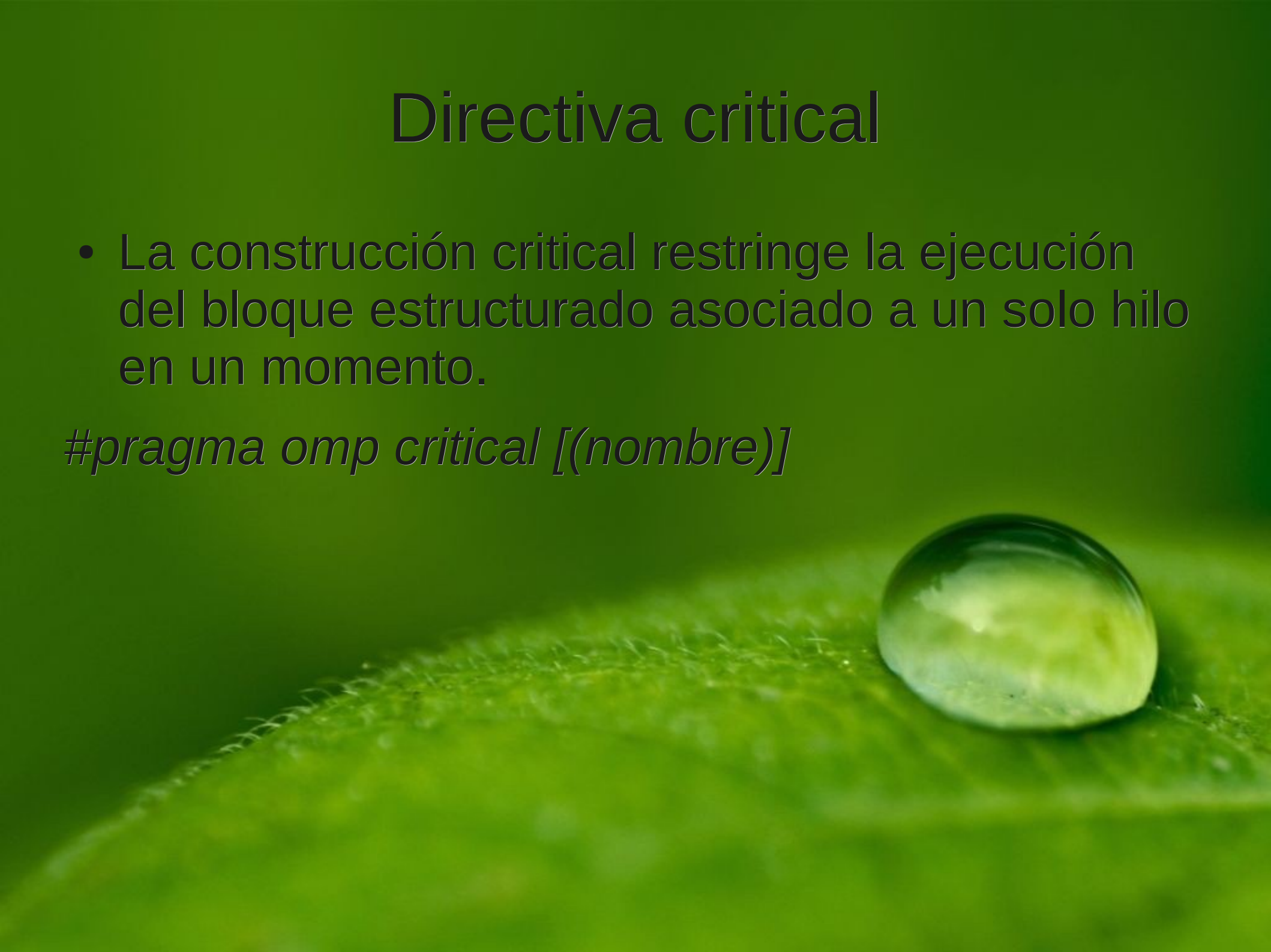
- La construcción \textbf{barrier} especifica una barrera explícita en el punto en el cual la construcción aparece.

#pragma omp barrier

Directiva critical

- La construcción critical restringe la ejecución del bloque estructurado asociado a un solo hilo en un momento.

#pragma omp critical [(nombre)]



Directiva atomic

- La construcción *atomic* asegura que una locación específica es actualizada automáticamente, en vez de exponerla a posibles hilos que escriban sobre el de manera múltiple o simultáneo.

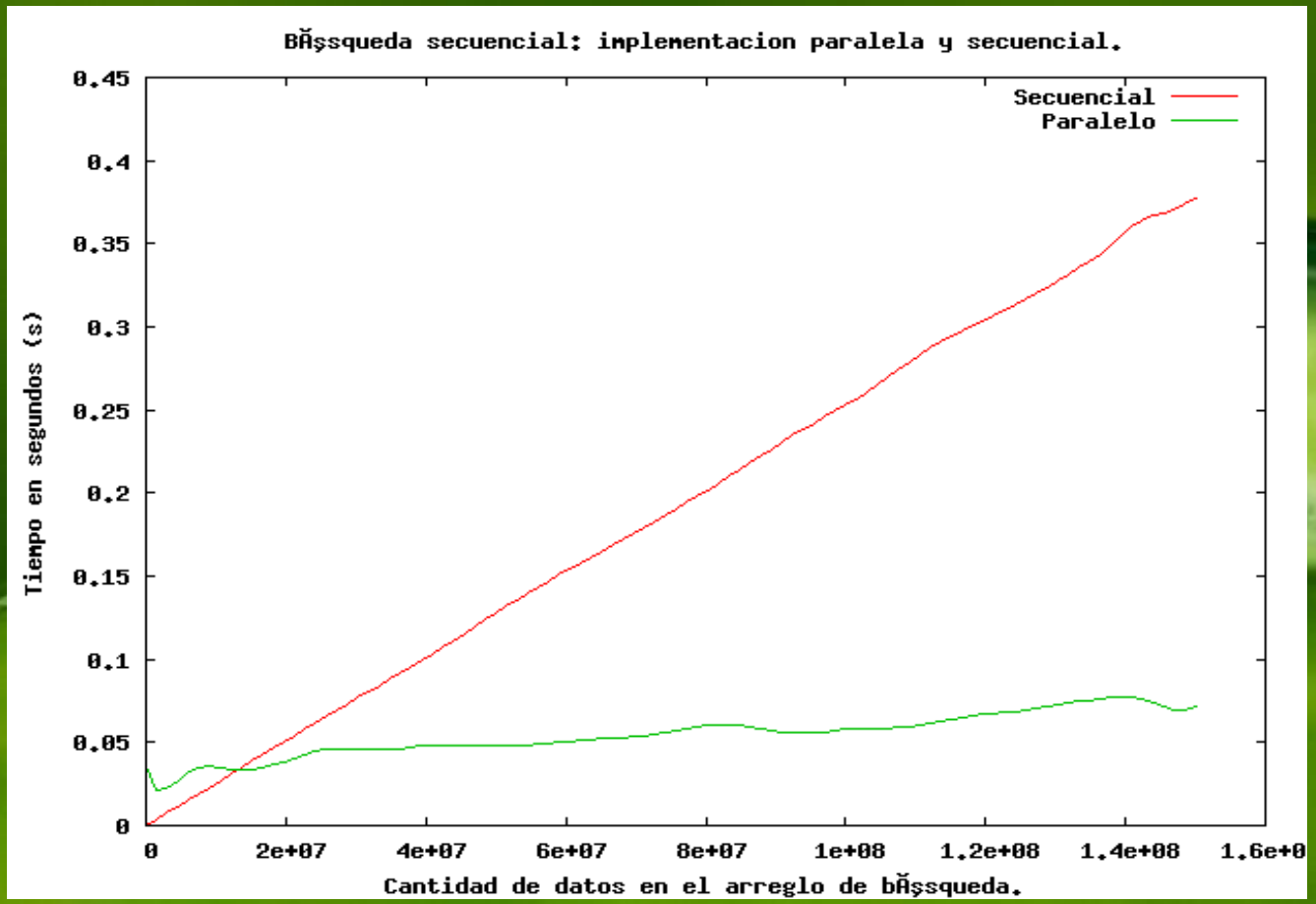
#pragma omp atomic [read | write | update | capture]

Ayuda a la hora de compilar: make

- Antes de comenzar a utilizar OpenMP, crearemos un sistema de producción con Autotools, que nos permitirá compilar nuestro código en nuestro sistema operativo, sin importar cual sea, y crear una fácil distribución del mismo para ser compilado en un sinfín de plataformas.

Rendimiento

- Es posible obtener mejoras significativas al renidmiento en procesamientos de datos muy extensos, mientras que se obtiene rendimientos inferiores con pocos datos. Esto lo muestra la siguiente gráfica de una búsqueda secuencial:



Aloritmos propuestos para desarrollar en el taller.

- Multiplicación vector por matriz. mxv
- Multiplicación de dos matrices
- Calcular la integral definida de una función dada.
- Regresión lineal.

Conclusiones.

- Los algoritmos paralelos son más afectivos con grandes cantidades de datos, y son inferiores con datos reducidos.
- La programación paralela varía en su rendimietno debido al entorno de tiempo de ejecución.
- Una importante limitación es la cantidad de memoria disponible.
- En un mundo donde los procesadores multicore reinan, es necesario comenzar a programar en paralelo.
- Para esto, OpenMP es una exceletne herramienta.


¿Alguna pregunta?