

Multiprocesamiento en lenguaje C

Introducción a

Open Multiprocessing (OpenMP)

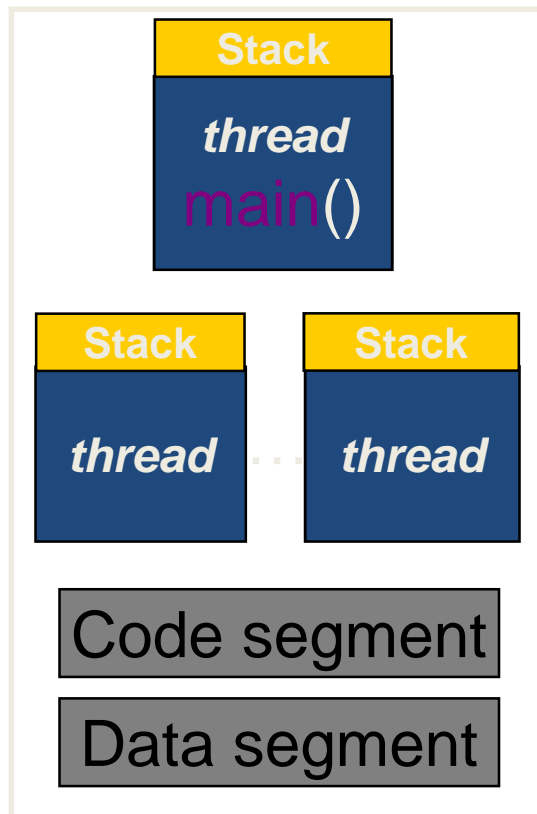
Proyecto PAPIME PE104911

Pertinencia de la enseñanza del
cómputo paralelo en el currículo de las
ingenierías

Algunas preguntas

- ¿Qué es un proceso?
- ¿Qué es un hilo?
- ¿Qué se comparte entre los hilos?
- ¿Cuál es la diferencia entre hilo y proceso?

Procesos e Hilos

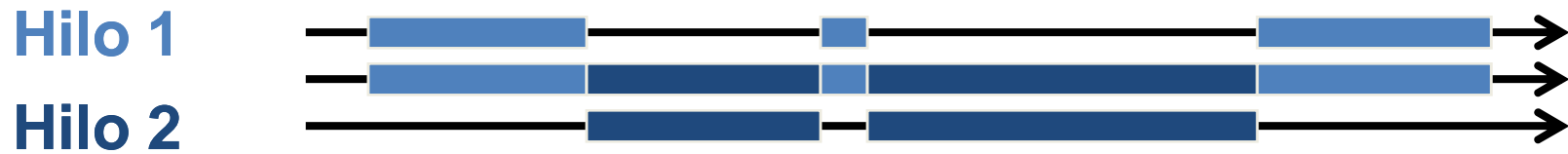


- Un proceso inicia ejecutando su punto de entrada como un hilo
- Los hilos pueden crear otros hilos dentro del proceso
 - Cada hilo obtiene su propio stack
- Todos los hilos dentro de un proceso **comparten código y segmentos de datos**

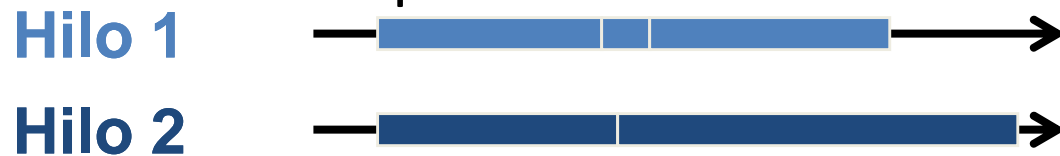
Conceptos importantes que se
deben recordar

Concurrencia vs. Paralelismo

- Concurrencia: dos o más hilos están en progreso al mismo tiempo:



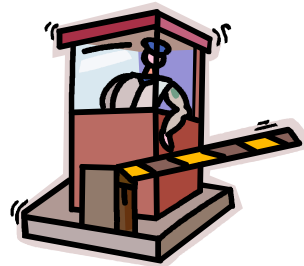
- Paralelismo: dos o más hilos están en ejecución al mismo tiempo



- Se requieren varios núcleos

Conceptos Importantes

- Condiciones de carrera (Data Race)
- Región Crítica
- Exclusión Mutua
- Sincronización
- Deadlock



Empezando con OpenMP

¿Qué es OpenMP?

- Modelo de programación paralelo.
- Paralelismo de memoria compartida.
- Extensiones para lenguajes de programación existentes (C, C++, Fortran)
- Combina código serial y paralelo en un solo archivo fuente.

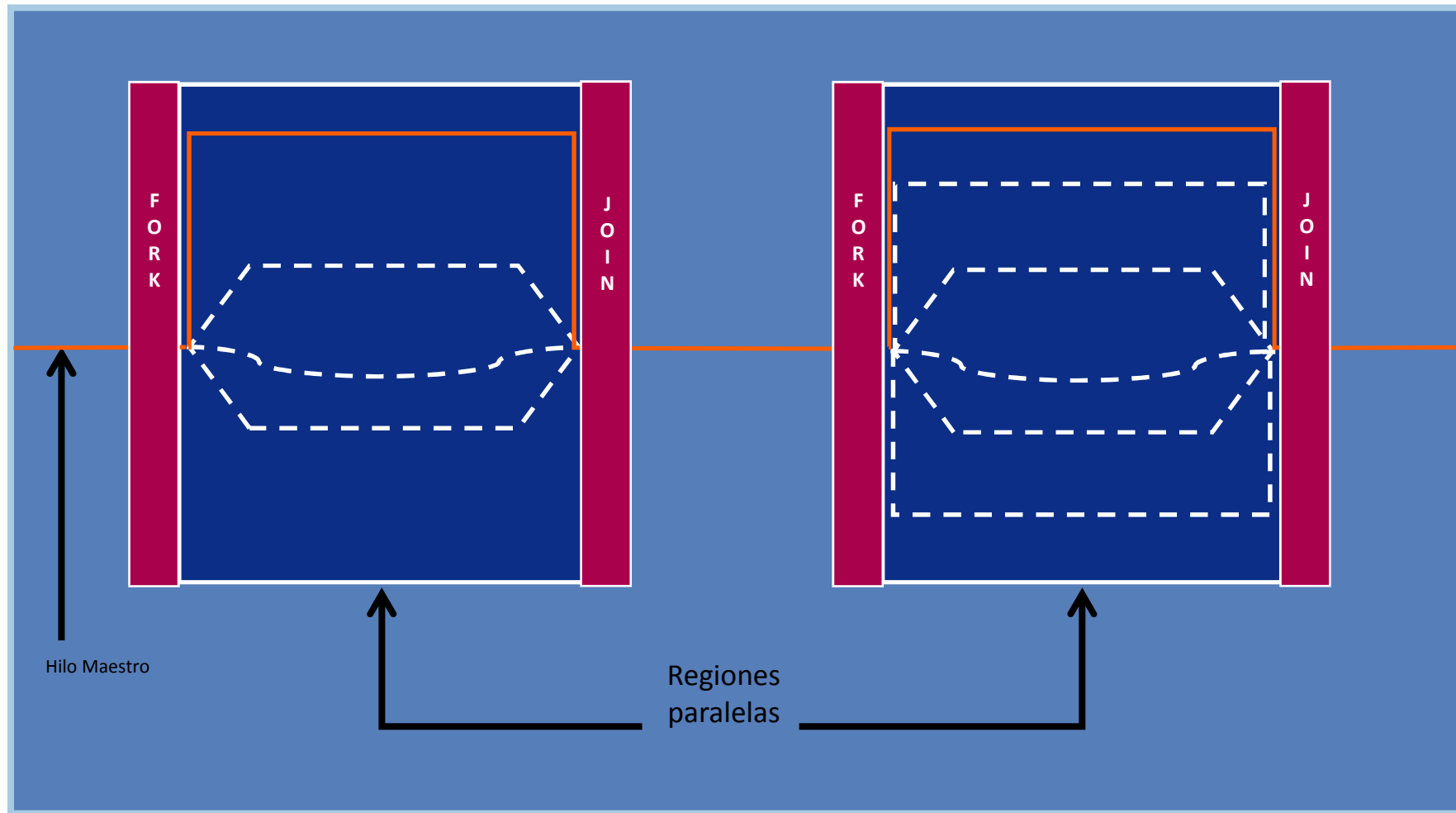
¿Qué es OpenMP?

- Conjunto de
 - directivas del compilador
 - bibliotecas de funciones
 - variables de ambiente,No un lenguaje.
- Basado en el modelo de hilos.

Arquitectura de OpenMP

- Fork/join (Maestro esclavo)
- Trabajo Y Datos Compartido entre hilos
- Maneja sincronización (barreras, otras)

Fork / Join

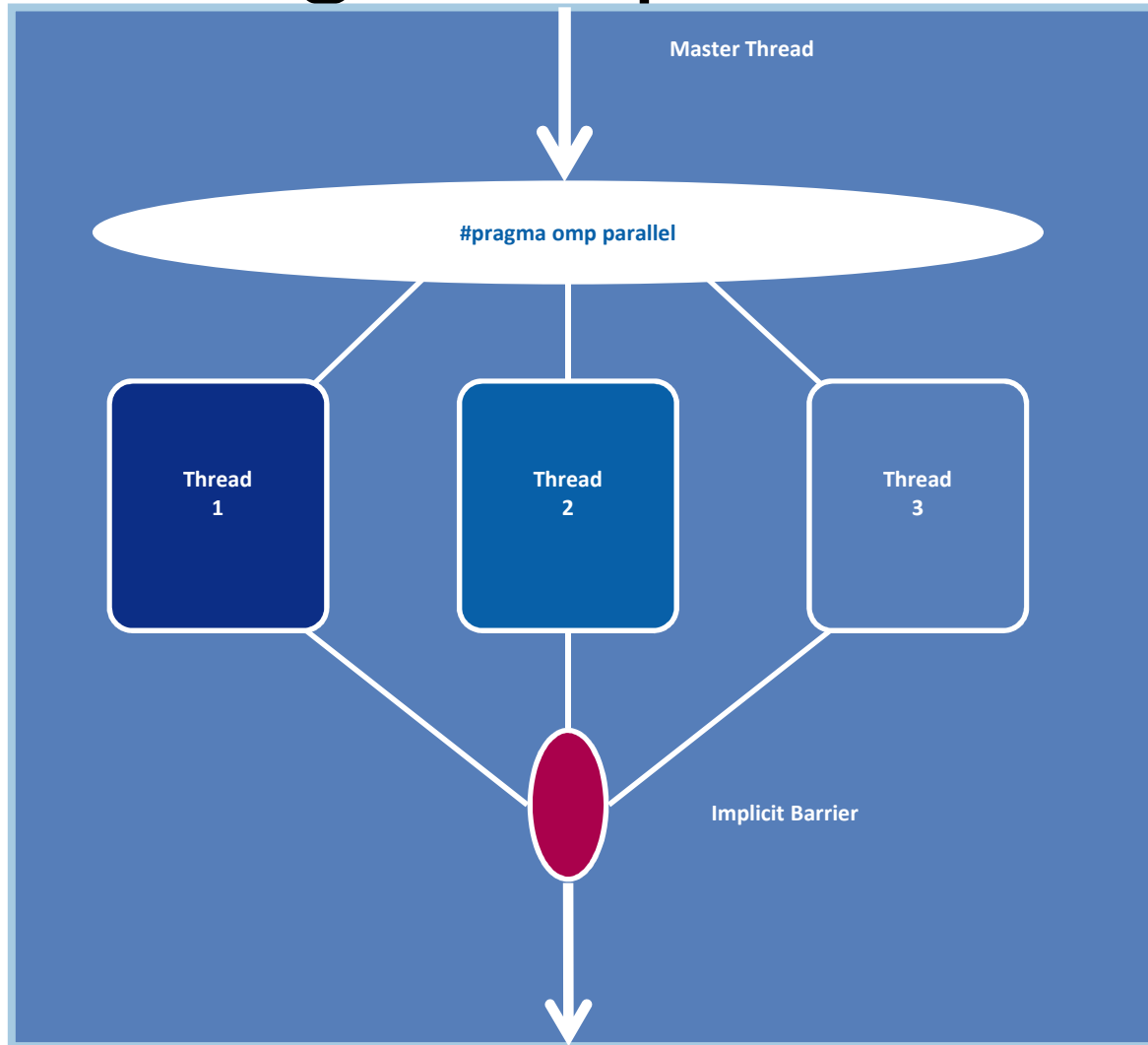


Sintaxis

- Directivas o pragmas
 - `#pragma omp construct [clause [clause]...]`
 - *Clausulas: Especifican atributos para compartir datos y calendarización*

Una pragma en C o C++ es un directivo al compilador.

Regiones paralelas




Regiones paralelas

- Los hilos son creados desde el pragma *parallel*.
- Los datos son compartidos entre los hilos.

C/C++ :

```
#pragma omp parallel  
{  
    block  
}
```



¿Cuántos hilos?

- Num. Hilos = Num. Procesadores o núcleos
- Intel lo usa de esta forma.
- Se definen más hilos con la variable de ambiente

OMP_NUM_THREADS.

Actividad 1

- Compilar la versión serial.

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

```
int main() {  
    int i;  
    printf("Hola Mundo\n");  
    for(i=0;i<6;i++)  
        printf("Iter:%d\n",i);  
  
    printf("Adios \n");  
}
```


Actividad 1

- Agregar la directiva para ejecutar las primeras cuatro líneas del main en paralelo.
- Compilar con la opción (-openmp)/Qopenmp
- ¿Qué sucede?

Actividad 2

- Aumentar el número de hilos a 4 y ejecutar el programa
- `$ export OMP_NUM_THREADS=4`
- Ejecutar el programa con más hilos y describir lo que sucede.

Work – Sharing constructor for

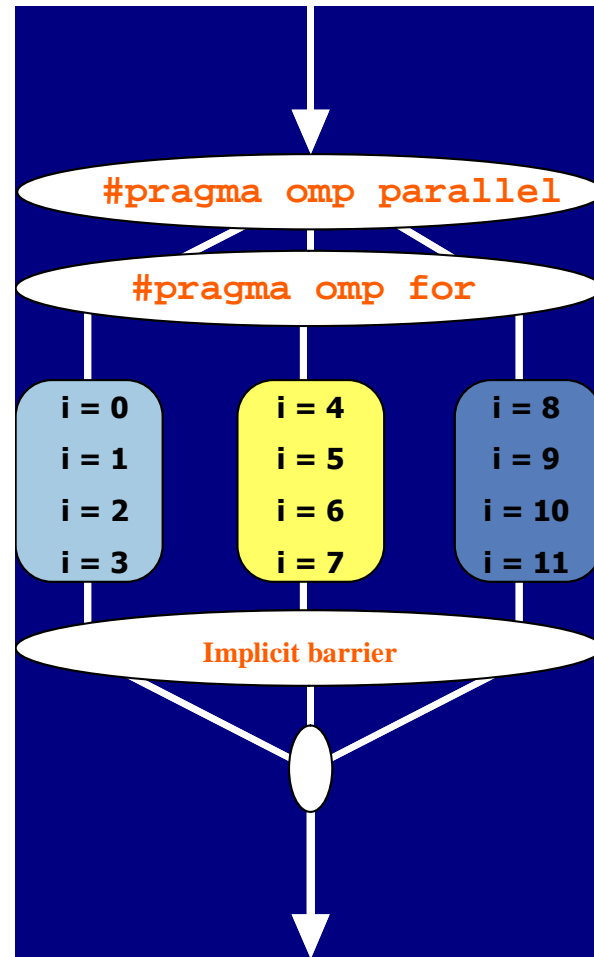
```
#pragma omp parallel
```

```
#pragma omp for
```

```
for(i=0;i<100;i++) {  
    Realizar Trabajo();  
}
```

- Divide los ciclos de la iteración entre los hilos.
- Debe estar especificada en una región paralela
- Debe estar antes del ciclo.

```
#pragma omp parallel
#pragma omp for
  for(i = 0; i < 12; i++)
    c[i] = a[i] + b[i]
```



Combinando Pragmas

- Estos dos segmentos de código son equivalentes.

```
#pragma omp parallel
{
    #pragma omp for
    for (i=0; i< MAX; i++) {
        res[i] = huge();
    }
}
```

```
#pragma omp parallel for
    for (i=0; i< MAX; i++) {
        res[i] = huge();
    }
```

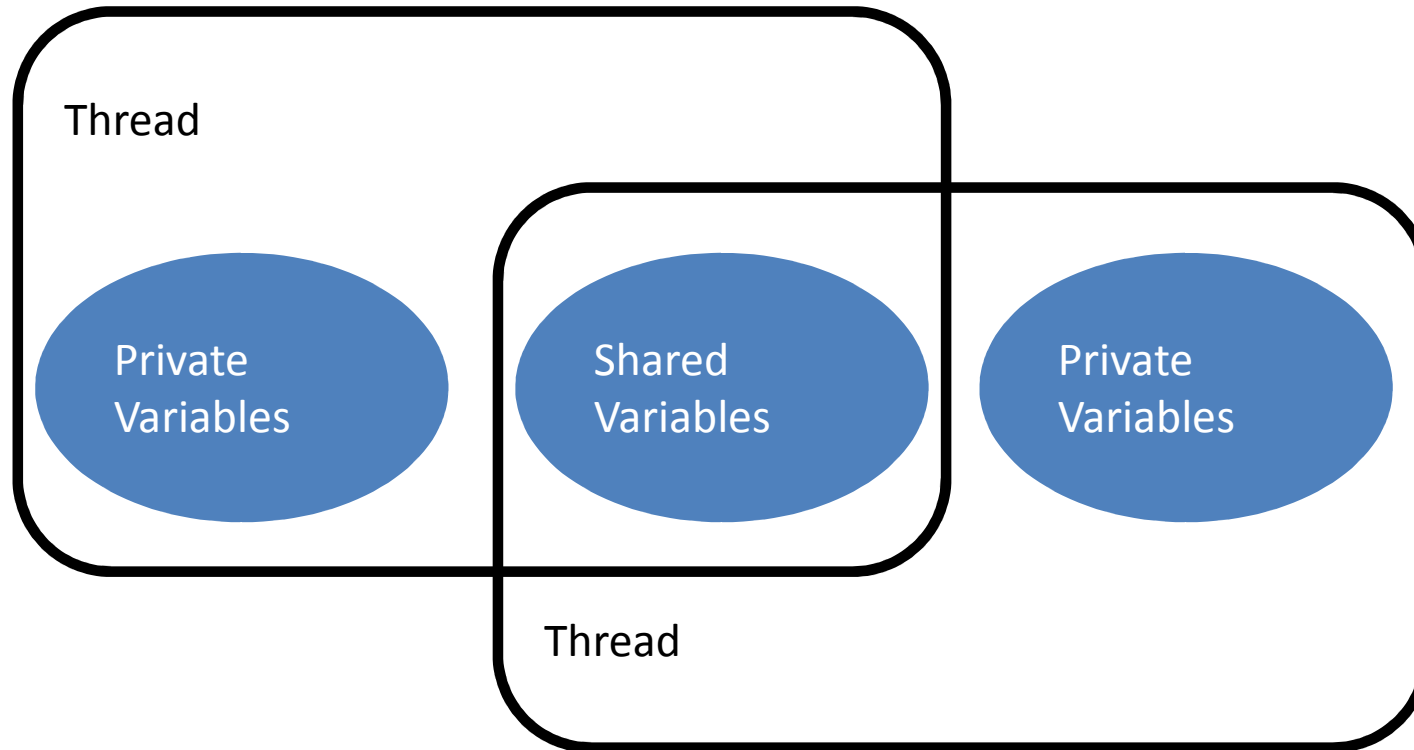
Actividad

- Modificar el programa anterior para dividir el número de iteraciones entre los hilos.

Funciones de ambiente

- `void omp_set_num_threads(int nthreads)`
- `int omp_get_num_threads(void)`
- `int omp_get_max_threads(void)`
- `int omp_get_thread_num(void)`
- `int omp_get_num_procs(void)`

Variables Compartidas y Privadas



Ejemplo Descomposición Dominio

Sequential Code:

```
int a[1000], i;  
for (i = 0; i < 1000; i++)  
    a[i]=foo(i);
```

Thread 0:

```
for (i=0; i<500; i++) a[i] = foo(i);
```

Thread 1:

```
for (i=500; i<1000; i++) a[i] = foo(i);
```

Private

Shared

Descomposición Funcional

- `volatile int e;`
 - `main () {`
 - `int x[10], j, k, m; j = f(x, k); m = g(x.`
 - `k);`
 - `}`
- Variables locales a funciones: Private

```
• int f(int *x, int k)                                Thread 0
• {
• int a; a = e * x[k] * x[k]; return a;
• }
```

```
• int g(int *x, int k)                                Thread 1
• {
• int a; k = k-1; a = e / x[k]; return a;
• }
```

Atributos de alcance

Clausulas shared y private

– `shared (varname, ...)`

– `private (varname, ...)`

Clausula private

- Realiza una copia de la variable para cada hilo.
 - Las variables no se inicializan
 - Cualquier valor externo a la región paralela se coloca como indefinido.

```
void* work(float* c, int N) {  
    float x, y; int i;  
    #pragma omp parallel for  
    private(x,y)  
        for(i=0; i<N; i++) {  
            x = a[i]; y = b[i];  
            c[i] = x + y  
        }  
}
```

Actividad

- Realizar un programa que sume dos arreglos unidimensionales y los almacene en un tercer arreglo.
- Versión Serial
- Versión Paralela (OPENMP)
 - Indicar que tipo de descomposición se utiliza.
 - Indicar con las con las clausulas Shared y Private cuales son privadas y compartidas.

Problemas con private

```
float prod_punto(float* a, float* b, int N)
{
    float sum = 0.0;
    #pragma omp parallel for
    for (int i=0; i<N; i++)
    {
        sum += a[i] * b[i];
    }
    return sum;
}
```

Cláusula reduce

La cláusula reduce funciona de manera similar a al MPI_Reduce.
(Cálculo de operaciones parciales y colectivas)

//Sintaxis: #pragma omp reduction(operator:variable)

¿Que hace?

```
#pragma omp parallel for reduction(+:sum)
```

```
for (i=0; i<N; i++)  
{  
    sum += a[i] * b[i];  
}
```

Operaciones con reduce (C/C++)

Operator	Initial Value
+	0
*	1
-	0
^	0

Operator	Initial Value
&	0
	0
&&	1
	0

Ejemplo

- Cálculo del número Pi
- $\pi = \int_0^1 \frac{4}{1+x^2} dx$
- La Regla de rectángulo consiste de estimar el área debajo de la curva $y=4/(1+x^2)$ entre $x=0$ y $x=1$ mediante áreas de rectángulos

Ejemplo

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

long long num_steps =
    1000000000;
double step;

int main(int argc, char* argv[])
{
    double x, pi, sum=0.0;
    int i;
    for (i=0; i<num_steps; i++)
    {
        x = (i + .5)*step;
        sum = sum + 4.0/(1.+ x*x);
    }

    pi = sum*step;

    printf("El valor de Pi es
    %15.12f\n",pi);
    return 0;
}
```

Actividad

- Paralelizar el código anterior, utilizando los constructores y cláusulas vistas

Constructor critical

```
float dot_prod(float* a, float* b, int N)
{
    float sum = 0.0;
    #pragma omp parallel for
    for (int i=0; i<N; i++)
    {
        #pragma omp critical
        sum += a[i] * b[i];
    }
    return sum;
}

// Sintaxis: #pragma omp critical [(lock_name)]
```

¿Cuál será el problema aquí?

Ejercicio- paralelizar

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

#define SIZE 10

int main()
{
    int i;
    int max;
    int a[SIZE];

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

    max = a[0];

    for (i = 1; i < SIZE; i++)
    {
        if (a[i] > max)
        {
            max = a[i];
        }
    }

    printf_s("max = %d\n", max);
}
```

Asignando Iteraciones

- La cláusula **schedule** permite dividir las iteraciones de los ciclos entre los hilos, indica como las iteraciones se asignan a los hilos.

- *schedule(static,[chunk])*
 - Bloques de iteraciones de tamaño “*chunk*” a los hilos
 - Distribución Round robin
- *schedule(dynamic,[chunk])*
 - Los hilos toman un numero “*chunk*” de iteraciones
- *schedule(guided,[chunck])*
 - Inicia con un bloque grande
 - El tamaño del bloque se hace mas pequeño, hasta llegar al tamaño del “*chunk*”

Recomendaciones de uso

Scheduling Clause	When to Use
Static	Predictable and similar work per iteration
Dynamic	Unpredictable, highly variable work per iteration
Guided	Special case of dynamic to reduce scheduling overhead

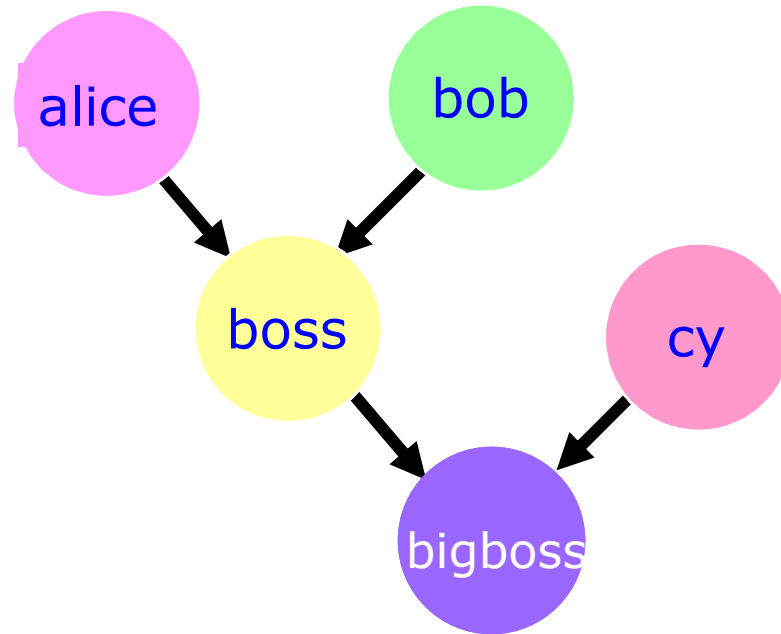
Ejemplo

```
#pragma omp parallel for schedule (static, 8)  
for( int i = start; i <= end; i += 2 )  
{  
    if ( TestForPrime(i) ) gPrimesFound++;  
}
```

Descomposicion de Tareas

```
a = alice();  
b = bob();  
s = boss(a, b);  
c = cy();  
printf ("%6.2f\n",  
        bigboss(s,c));
```

Descomposicion de Tareas

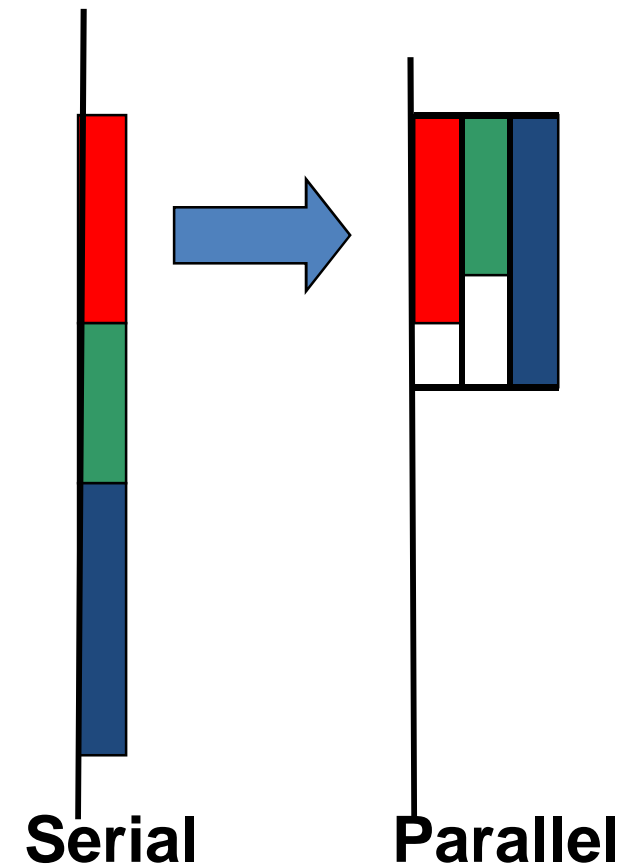


Paralelismo funcional

Secciones Paralelas

- Secciones independientes de código, se pueden ejecutar de forma concurrente

```
#pragma omp parallel sections
{
    #pragma omp section
    phase1();
    #pragma omp section
    phase2();
    #pragma omp section
    phase3();
}
```



Constructor single

- #pragma omp single

```
#pragma omp parallel
{
    DoManyThings();
    #pragma omp single
    {
        ExchangeBoundaries();
    } // Hilos esperan

    DoManyMoreThings();
}
```

Constructor maestro

```
#pragma omp master {  
  
    #pragma omp parallel  
    {  
        DoManyThings();  
        #pragma omp master  
        { // si no es el maestro, salta  
            ExchangeBoundaries();  
        }  
        DoManyMoreThings();  
    }  
}
```

Barreras Implícitas

- Algunos constructores tiene barreras implícitas
 - Parallel
 - For
 - Single
- Barreras no necesarias perjudican el desempeño

Clausula nowait

- Permite ignorar barreras implícitas

```
#pragma omp for nowait
for(...)
{...};
```

```
#pragma single nowait
{ [...] }
```

```
#pragma omp for schedule(dynamic,1) nowait
for(int i=0; i<n; i++)
    a[i] = bigFunc1(i);

#pragma omp for schedule(dynamic,1)
for(int j=0; j<m; j++)
    b[j] = bigFunc2(j);
```


Constructor Barrier

- Barrera Explicita
- Cada hilo espera hasta que todos lleguen a la barrera

```
#pragma omp parallel shared (A, B, C)
{
    DoSomeWork(A,B);
    printf("Processed A into B\n");
#pragma omp barrier
    DoSomeWork(B,C);
    printf("Processed B into C\n");
}
```

Ejercicio

- Paralelizar los dos códigos proporcionados, utilizando los constructores y cláusulas vistos.

Referencias

- Introduction to parallel Programming ,Student book , Intel Software College , 2007
- Multicore Programming for Academia, Intel Software College, 2007
- <https://computing.llnl.gov/tutorials/openMP/>