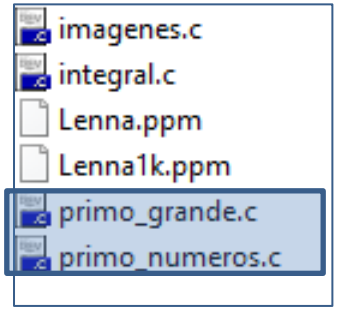
Práctica 1

Sesión 3

Objetivos

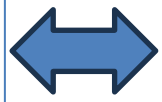
- Paralelizar códigos mediante el identificador de hilo y el número de hilos en una región paralela
- Paralelizar dos códigos secuenciales que calculen:
  1. El mayor número primo del tipo entero sin signo de 8 bytes (**unsigned long long**)
  2. El número de nº primos menores o iguales que un cierto entero positivo dado (**primo_números.c**)

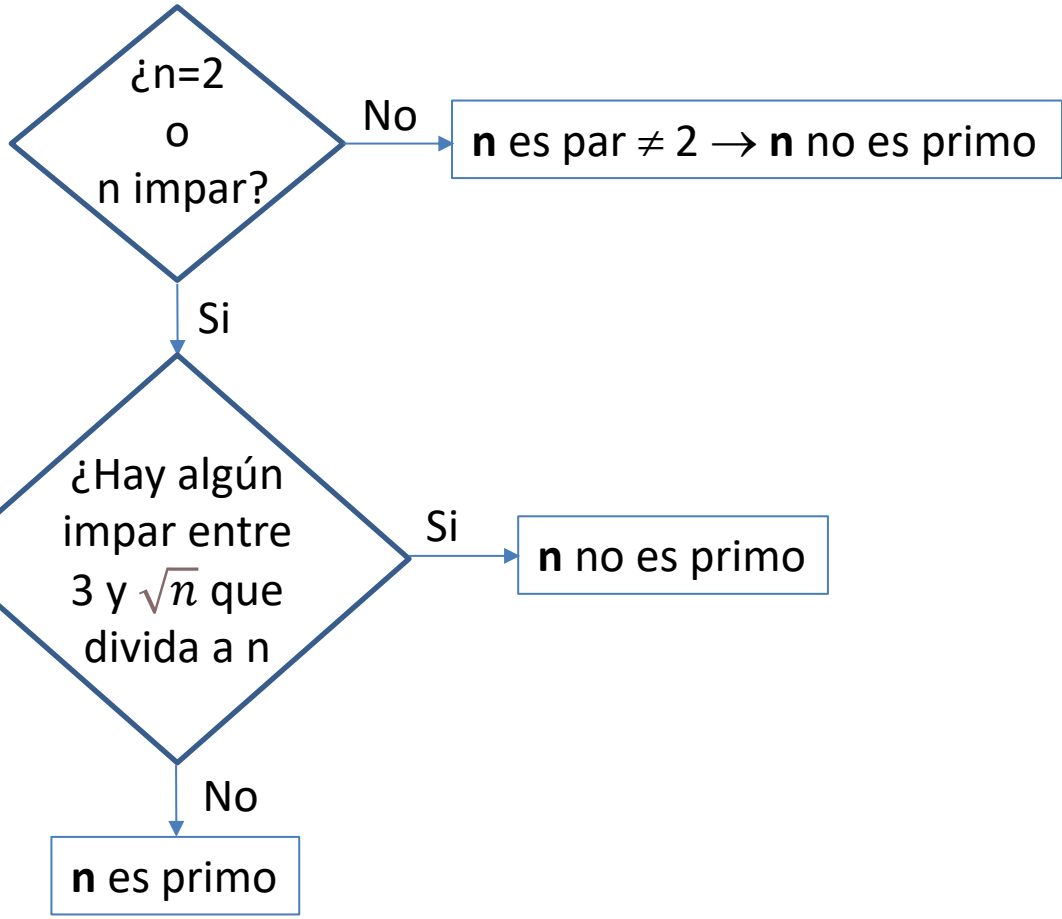


Ejercicio 1

Algoritmo: Determinar si un número es primo (función **primo**)

```
int primo(Entero_grande n){
   int p;
   Entero_grande i, s;
   p = (n % 2 != 0 || n == 2);
   if (p) {
        s = sqrt(n);
        for (i = 3; p && i <= s; i += 2)
              if (n % i == 0) p = 0;
    }
    return p;
}
```





Ejercicio 1:

- Paralelizar el código primo_grande.c: calcula el mayor número primo del tipo entero sin signo de 8 bytes (unsigned long long)

```c
#include <stdio.h>
#include <math.h>
#include <limits.h>
typedef unsigned long long Entero_grande;
#define ENTERO_MAS_GRANDE  ULLONG_MAX
int primo(Entero_grande n){
  int p;
  Entero_grande i, s;
  p = (n % 2 != 0 || n == 2);
  if (p) {
    s = sqrt(n);
    for (i = 3; p && i <= s; i += 2)
      if (n % i == 0) p = 0;
  }
  return p;
}

int main(){
  Entero_grande n;
  for (n = ENTERO_MAS_GRANDE; !primo(n); n -= 2) {
    /* NADA */
  }
  printf("El mayor primo que cabe en %lu bytes es %llu.\n",
         sizeof(Entero_grande), n);
  return 0;
}
```

Paralelizar el bucle

Nota: ENTERO_MAS_GRANDE=ULLONG_MAX=18446744073709551615

Ejercicio 1

Paralelizar el código de la función **primo** incluida en primo_grande.c:

Código secuencial a paralelizar en función primo:

```
for (i = 3; p && i <= s; i += 2)
    if (n % i == 0) p = 0;
```

Código paralelo 1:

```
#pragma omp parallel for
for (i = 3; p && i <= s; i += 2)
    if (n % i == 0) p = 0;
```

→ Falla al compilar. ¿Porqué?

```
pprimo_grande.c: In function 'primo':
pprimo_grande.c:19:17: error: invalid controlling predicate
     for (i = 3; p && i <= s; i += 2)
                 ^
```

Código paralelo 2:

```
#pragma omp parallel for
for (i = 3;i <= s; i += 2)
    if (n % i == 0) p = 0;
```

→ Funciona, pero es ineficiente (¿porqué?)

Ejercicio 1

Paralelizar el código de la función **primo** incluida en primo_grande.c. Copia primo_grande.c con nombre pprimo_grande.c y modifícalo:

Repartir las iteraciones de acuerdo al identificador del hilo:

```
#pragma omp parallel ……..//poner claúsulas
{
   int hilo = omp_get_thread_num();
   int nhilos = omp_get_num_threads();
   for (i =¿?; p && i <= s; i += ¿?)
        if (n % i == 0) p = 0;
}
```

cambiar → `for (i = vi; p && i <= s; i += inc)`

Una forma sencilla de hacerlo consiste en realizar un reparto cíclico de los índices:

Ejemplo: repartir 3-5-7-9-11-13-15-17-19-21 con 3 hilos:

- H0: 3, 9, 15, 21
- H1: 5, 11, 17
- H2: 7, 13, 19

Ejemplo: repartir 3-5-7- 9 -11-13-15-17-19-21 con 4 hilos:

- H0: 3, 11, 19
- H1: 5, 13, 21
- H2: 7, 15
- H3: 9, 17

Tendrás que determinar el valor inicial (vi) y el valor del incremento (inc) en función del número de hilos nhilos y del identificador del hilo

Ejercicio 1

```
#pragma omp parallel ……..//poner claúsulas
{
   int hilo = omp_get_thread_num();
   int nhilos = omp_get_num_threads();
   for (i = vi; p && i <= s; i += inc)
        if (n % i == 0) p = 0;
}
```

Debes sustituir vi e inc con expresiones que dependan de hilo y nhilos

Ejemplo: repartir 3-5-7-9-11-13-15-17-19-21 con 3 hilos:

- H0: 3, 9, 15, 21
- H1: 5, 11, 17
- H2: 7, 13, 19

Ejemplo: repartir 3-5-7- 9 -11-13-15-17-19-21 con 4 hilos:

- H0: 3, 11, 19
- H1: 5, 13, 21
- H2: 7, 15
- H3: 9, 17

hilo	vi=¿?
0	3
1	5
2	7
3	9

nhilos	inc=¿?
3	6
4	8
5	10
6	12

Notas

- Es conveniente declarar la variable p como volátil:

  volatile int p;

- El modificador volatile del lenguaje C le indica al compilador que no optimice el acceso a esa variable (que no la cargue en registros y que cualquier acceso a ella se haga efectivamente sobre memoria), con lo que la modificación de su valor por parte de un hilo será “visible antes” para el resto de los hilos

Ejercicio 1

Compila el código secuencial y el código paralelo, comprobando que el código paralelo funciona correctamente

```
-bash-4.1$ gcc -o primo_grande primo_grande.c -lm
-bash-4.1$ ./primo_grande
El mayor primo que cabe en 8 bytes es 18446744073709551557.
-bash-4.1$ gcc -fopenmp -o pprimo_grande pprimo_grande.c -lm
-bash-4.1$ ./pprimo_grande
Tiempo (4): 5.242235
El mayor primo que cabe en 8 bytes es 18446744073709551557.
```

Ejercicio 2

Paralelizar contar número de primos (primos_numero.c)

```c
#include <stdio.h>
#include <math.h>
typedef unsigned long long Entero_grande;
#define N 100000000ULL

int primo(Entero_grande n){
  int p;
  Entero_grande i, s;
  p = (n % 2 != 0 || n == 2);
  if (p) {
    s = sqrt(n);
    for (i = 3; p && i <= s; i += 2)
      if (n % i == 0) p = 0;
  }
  return p;
}

int main(){
  Entero_grande i, n;
  n = 2; /* Por el 1 y el 2 */
  for (i = 3; i <= N; i += 2)
    if (primo(i)) n++;
  printf("Entre el 1 y el %llu hay %llu numeros primos.\n", N, n);
  return 0;
}
```

Ejercicio 2: Contar número de primos (primos_numero.c)

- 1ª Opción

  Aprovecha la implementación paralela de primo (pprimo) del ejercicio 1:

  ¡¡no es eficiente!!

  ¿Porqué?

  ……

```
int pprimo(Entero_grande n){  //Implementación paralela del ejercicio 1
……..
}
int main(){
   Entero_grande i, n;
   n = 2; /* Por el 1 y el 2 */
   for (i = 3; i <= N; i += 2)
   if (pprimo(i)) n++;
        printf("Entre el 1 y el %llu hay %llu numeros primos.\n",N, n);
   return 0;
}
```

Ejercicio 2: Contar número de primos (primos_numero.c)

- 2ª Opción
  - Paralelizar el bucle del programa principal, usando la versión secuencial de la función **primo**

```
……
int primo(Entero_grande n){   //Implementación secuencial del ejercicio 1
…….
}
int main(){
  Entero_grande i, n;
  n = 2; /* Por el 1 y el 2 */
   #pragma omp parallel for ……(añadir claúsulas)
  for (i = 3; i <= N; i += 2)
      if (primo(i)) n++;
   printf("Entre el 1 y el %llu hay %llu numeros primos.\n",N, n);
  return 0;
}
```

¿Las iteraciones tienen el mismo coste computacional?

Ejercicio 2

- Como el coste computacional de las iteraciones es diferente, menor en las iteraciones inferiores, es conveniente modificar la planificación por defecto (static,0)
- Prueba con distintas planificaciones en tiempo real:
  - Añade la claúsula schedule(runtime) en el bucle paralelo for:

    #pragma omp parallel for **schedule(runtime)** …..
  - Al ejecutar el código paralelo usa la variable de entorno OMP_SCHEDULE. Por ejemplo,

    OMP_SCHEDULE=dynamic,2 ./miprograma (planificación dinámica con chunk igual 2)

    OMP_SCHEDULE=static,3 ./miprograma (planificación estática con tamaño de chunk igual a 3)

Ejercicio 2

- Para que se muestren el nº de hilos, la planificación y el tiempo de ejecución, puedes añadir en el programa principal el siguiente código:

```
double t = omp_get_wtime();
#pragma omp parallel for schedule(runtime) …..
 for (i = 3; i <= N; i += 2)
    if (primo(i)) n++;
t = omp_get_wtime() - t;
#pragma omp parallel
  #pragma omp single
    printf("\tTiempo  para %d hilos: %f\n",omp_get_num_threads(),t);
```

- El resultado que debe dar es: 5761456

Ejercicio 2

- Para ejecutar el programa en el cluster de Kahan, utiliza el siguiente script (**primos.sh**) y lánzalo al sistema de colas de Kahan, mediante el comando sbatch:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --time=5:00
#SBATCH --partition=cpa
# SBATCH -o primos.txt
echo "Código paralelo con planificación estática y chunk=0 (reparto por bloques)"
OMP_SCHEDULE=static,0 ./pprimo_numeros
echo "Código paralelo con planificación estática y chunk=1 (reparto cíclico de 1 en 1)"
OMP_SCHEDULE=static,1 ./pprimo_numeros
echo "Código paralelo con planificación estática y chunk=2 (reparto cíclico de 2 en 2)"
OMP_SCHEDULE=static,1 ./pprimo_numeros
echo "Código paralelo con planificación dinámica y chunk=1"
OMP_SCHEDULE=dynamic,1 ./pprimo_numeros
echo "Código paralelo con planificación dinámica y chunk=2"
OMP_SCHEDULE=dynamic,2 ./pprimo_numeros
```

Ejercicio 2

```
Código paralelo con planificación estática y chunk=0 (reparto por bloques)
Entre el 1 y el 100000000 hay 5761456 numeros primos.
        Tiempo  para 64 hilos: 6.909197

Código paralelo con planificación estática y chunk=1 (reparto cíclico de 1 en 1)
Entre el 1 y el 100000000 hay 5761456 numeros primos.
        Tiempo  para 64 hilos: 6.171748

Código paralelo con planificación estática y chunk=2 (reparto cíclico de 2 en 2)
Entre el 1 y el 100000000 hay 5761456 numeros primos.
        Tiempo  para 64 hilos: 6.181269

Código paralelo con planificación dinámica y chunk=1
Entre el 1 y el 100000000 hay 5761456 numeros primos.
        Tiempo  para 64 hilos: 6.175966

Código paralelo con planificación dinámica y chunk=2
Entre el 1 y el 100000000 hay 5761456 numeros primos.
        Tiempo  para 64 hilos: 6.126350
```