DESARROLLO DE SOFTWARE EN ARQUITECTURAS PARALELAS

1. Motivación y aspectos de la programación paralela.
2. Tipos de sistemas paralelos. Paradigmas de programación paralela.
3. Conceptos básicos y medidas de paralelismo.
4. Diseño de programas paralelos.
5. La Interface de paso de mensaje: el estándar MPI.
6. Paralelización de algoritmos: ejemplos y aplicaciones.
   - Listado de números primos.
   - Detección de contornos o aristas en una imagen.
   - Multiplicación matriz-matriz por bloques: algoritmo de Cannon.
   - Algoritmo de Floyd-Warshall para la obtención de los caminos más cortos en un grafo.

DSAP

Listado de números primos

¿Cuándo un entero positivo se dice que es primo?

Diremos que **p** $\in \mathbb{Z}^+$ es primo si tiene exactamente dos divisores positivos distintos.

- Los divisores positivos de 13 son 1 y 13. Luego 13 es primo.
- El entero 15 no solo tiene como divisores positivos el 1 y 15. Tiene otros divisores, el 3 y el 5. Luego 15 no es primo.
- El entero 1 solo tiene un divisor positivo: el propio 1. Luego 1 no es primo.

Los algoritmos existentes para determinar si un entero es primo tienen el suficientemente coste computacional como para poder incluir paralelismo.

Si a esto le añadimos que este cálculo se debe realizar muchas veces, el coste computacional se incrementa.

Listado de números primos

Problema a resolver:

Dado un entero positivo **n** $\in \mathbb{Z}^+$, determinar cuántos enteros entre 1 y **n**, son primos, es decir, el problema consiste en obtener el número de primos menores que **n**.

Algoritmo:

```
total_primos = 0;
for (int i = 2; i <= n; i++)
        if (esPrimo(i) == 1)
                total_primos++;
```

Donde esPrimo(i) devuelve 1 si el entero i es primo.

DSAP

Listado de números primos

Algoritmo:

```
total_primos = 0;
for (int i = 2; i <= n; i++)
        if (esPrimo(i) == 1)
                total_primos++;
```

Este algoritmo es un algoritmo básico para obtener todos los primos menores que **n**. Aunque existen otros algoritmos más eficientes (como la criba de Eratóstenes) no son de interés en nuestro contexto.

Antes de seguir con la paralelización de este algoritmo veamos cómo ver si un entero es primo, es decir, la función esPrimo(i):

Listado de números primos

El algoritmo básico para saber si un **entero m, mayor que 1**, es primo, busca posibles divisores de **m** entre 2 y sqrt(**m**).

Función esPrimo(·):

```
int esPrimo( int m ) {
        for (int i = 2; i <= sqrt(m); i++)
                if ( m%i == 0)
                        return 0;
        return 1;}
```

¿Por qué entre 2 y sqrt(m)?

Es suficiente buscar entre 2 y sqrt(**m**) ya que si **p**>sqrt(**m**) fuese un divisor de **m**, existirá otro divisor menor que sqrt(**m**):

Si m = p·d, con p>sqrt(m), entonces:

$$d = m/p < m/\sqrt{m} = \sqrt{m}$$

Listado de números primos

Paralelización del algoritmo:

```
1       total_primos = 0;
2       for (int i = 2; i <= n; i++)
3               if (esPrimo(i) == 1)
4                       total_primos++;
```

Si disponemos de un total de **nproc** procesos trabajando en paralelo, el algoritmo anterior puede ser fácilmente paralelizado teniendo en cuenta que los cálculos representados por el bucle (2) son independientes.

En consecuencia, lo único que debemos plantearnos es una estrategia para distribuir la carga de trabajo entre los procesos.

DSAP

Listado de números primos

```
1  total_primos = 0;
2  for (int i = 2; i <= n; i++)
3              if (esPrimo(i) == 1)
4                          total_primos++;
```

Estrategias de distribución:

- Distribución por bloques consecutivos
- Distribución cíclica
- Distribución cíclica por bloques

DSAP

Listado de números primos

```
1 total_primos = 0;
2 for (int i = 2; i <= n; i++)
3             if (esPrimo(i) == 1)
4                         total_primos++;
```

Distribución por bloques consecutivos

Cada proceso evalúa un conjunto consecutivo de enteros.

Por ejemplo, si n=100 y p=4:

- El proceso 0 evalúa los enteros 2,3,…,25.
- El proceso 1 evalúa los enteros 26,27,…,50.
- El proceso 2 evalúa los enteros 51,52,…,75.
- El proceso 3 evalúa los enteros 76,77,…,100.

P0				P1				P2				P3			
2	3	…	25	26	27	…	50	51	52	…	75	76	77	…	100

Problema: Es más costoso evaluar primos grandes que pequeños

DSAP

Listado de números primos

```
1 total_primos = 0;
2 for (int i = 2; i <= n; i++)
3             if (esPrimo(i) == 1)
4                         total_primos++;
```

Distribución cíclica

Los procesos evalúan de forma cíclica el bucle (2).

Por ejemplo, si n=100 y p=4:

- El proceso 0 evalúa los enteros 2,6,10,…
- El proceso 1 evalúa los enteros 3,7,11,…
- El proceso 2 evalúa los enteros 4,8,12,…
- El proceso 3 evalúa los enteros 5,9,13,…

PROBLEMA: ¿?

P0	P1	P2	P3	P0	P1	P2	P3	P0	P1	P2	P3	…
2	3	4	5	6	7	8	9	10	11	12	13	…

```
1 total_primos = 0;
2 for (int i = 2 + myrank; i <= n; i = i + nprocs)
3             if (esPrimo(i) == 1)
4                         total_primos++;
```

DSAP

Listado de números primos

Distribución cíclica por bloques

Los procesos evalúan bloques de un determinado tamaño (bloque) de forma cíclica.

Por ejemplo, si n=100, p=4 y bloque=2:

- El proceso 0 evalúa los enteros 2,3,10,11,…
- El proceso 1 evalúa los enteros 4,5,12,13,…
- El proceso 2 evalúa los enteros 6,7,14,15,…
- El proceso 3 evalúa los enteros 8,9,16,17,…

```
1  total_primos = 0;
2  for (int i = 2; i <= n; i++)
3              if (esPrimo(i) == 1)
4                          total_primos++;
```

P0		P1		P2		P3		P0		P1		P2		P3		...
2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...

```
1  total_primos = 0;
2  inicio = myrank*bloque + 2;
3  while (inicio <= n) {
4              for (int i = inicio; i < inicio+bloque & i < n; i++)
5                          if (esPrimo(i) == 1)
6                                      total_primos++;
7              inicio = inicio + nprocs*bloque; }
```