



# Programación Paralela - CC332

2021-I

---

José Fiestas

21 de mayo de 2021

Universidad Nacional de Ingeniería  
[jose.fiestas@uni.edu.pe](mailto:jose.fiestas@uni.edu.pe)

# Unidad 3. Descomposición en paralelo

Objetivos:

1. Paralelismo directo
2. Uso de random en paralelo
3. Particionamiento, Divide y Vencerás

## Paralelismo directo

---

# Paralelismo directo

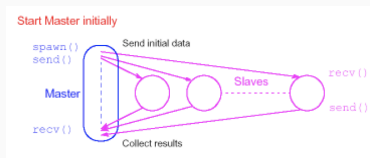
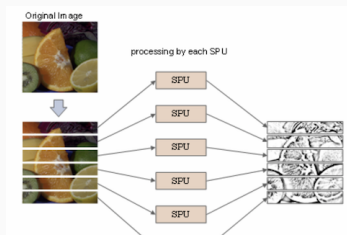
Donde

las tareas son completamente separables. Incluso no es necesaria la comunicación entre procesos

Ejemplo:

## procesamiento de imágenes

Solo las coordenadas del pixel son modificadas, independientemente de los pixels vecinos (desplazamiento, escala, rotación, etc)



Se toma especial cuidado en la distribución de la imagen entre procesos, normalmente se separa la imagen en regiones cuadradas, o en líneas y columnas.

# Paralelismo directo

Si se necesitan dos pasos de transformación para cada pixel:  $(x' = x + \Delta x; y' = y + \Delta y)$

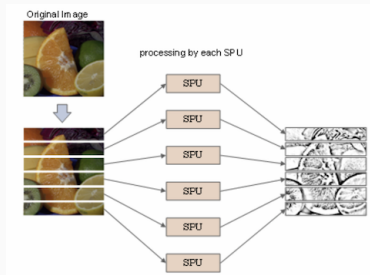
Tendremos una secuencia de cálculos de  $n \times n$  pixels en tiempo  $t_s = 2n^2$ , es decir  $O(n^2)$

Para  $p$  procesos, cada uno con 4 resultados (antiguas y nuevas coordenadas), es el tiempo de comunicación:

$$t_{comm} = p(t_{startup} + t_{data} + n^2(t_{startup} + 4t_{data})) = O(p + n^2)$$

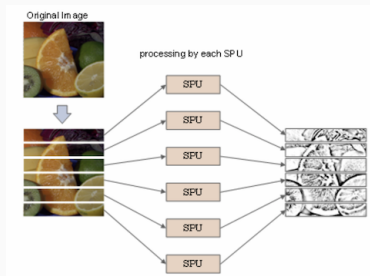
$$\text{Y el tiempo de cómputo: } t_{comp} = 2\frac{n^2}{p} = O(\frac{n^2}{p})$$

$$\text{Por consiguiente: } t_p = t_{comm} + t_{comp} = O(n^2)$$



# Paralelismo directo

Asimismo,  
el ratio cómputo/comunicación  
es constante,  $O(\frac{n^2/p}{p+n^2})$  si  $p$  es  
constante. Ya que la comunicación  
a los procesos y el retorno  
es lo que más cuesta, arquitectura  
de memoria compartida es óptima  
para este tipo de problemas.

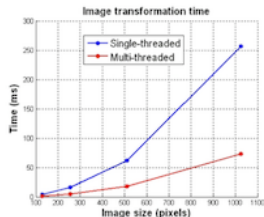


## Ejemplo: **Image wrapping**:

Transformación de giro de una imagen con respecto a un centro de rotación  $(c_x, c_y)$

$$x' = (x - c_x)\cos\theta + (y - c_y)\sin\theta + c_x$$

$$y' = (y - c_y)\sin\theta + (y - c_y)\cos\theta + c_y$$



## Ejemplo: Image wrapping:

Listing 1: Parallelized code for the image warp.

```
int index, xp, yp, tx = width / 2, ty = height / 2;
float x, y, radius, theta, PI = 3.141527f, DRAD = 180.0f / PI;
#pragma omp parallel for \
shared(inputImage, outputImage, width, height) \
private(x, y, index, radius, theta, xp, yp)
for (yp = 0; yp < height; yp++) {
    for (xp = 0; xp < width; xp++) {
        index = xp + yp * width;
        radius = sqrtf((xp - tx) * (xp - tx) + (yp - ty) * (yp - ty));
        theta = (radius / 2) * DRAD;
        x = cos(theta) * (xp - tx) - sin(theta) * (yp - ty) + tx;
        y = sin(theta) * (xp - tx) + cos(theta) * (yp - ty) + ty;
        outputImage[index] = BilinearlyInterpolate(inputImage, width, height, x, y);
    }
}
```



Ejemplo: **DFT (Transformada discreta de Fourier)**:

Dada la secuencia  $x_0, x_1, \dots, x_{n-1}$ , se calcula la secuencia

$y_0, y_1, \dots, y_{n-1}$ , según  $y_i = \sum_{j=0}^{n-1} w_n^{ij} x_j$

O en forma matricial,  $y = F_n x$ :

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)^2} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

Y ya que  $w_n^{ij}$  puede ser un número imaginario, se expresa como

$$w_n^j = e^{2\pi j i / n} = \cos(2\pi j / n) + i \sin(2\pi j / n)$$

Ejemplo: **DFT (Transformada discreta de Fourier)**:

Asimismo, para recuperar la secuencia  $x$ , se calcula

$$x_i = \frac{1}{n} \sum_{j=0}^{n-1} w_n^{-ij} y_j$$

Note que  $n$  procesos, pueden hacer este calculo en  $O(n)$  , sin necesidad de comunicación entre procesos  $j$ . Además , cualquier algoritmo de multiplicación matriz- vector puede solucionar el problema

# Paralelismo directo

Ejemplo: **FFT (Transformada rápida de Fourier)**:

Si particionamos la sumatoria de DFT en índices pares e impares, obtenemos

$$\begin{aligned} y_i &= \sum_{j=0}^{n-1} w_n^{ij} x_j = \sum_{j \text{ par}} w_n^{ij} x_j + \sum_{j \text{ impar}} w_n^{ij} x_j \\ &= \sum_{r=0}^{n/2-1} w_{n/2}^{ir} x_{2r} + w_n^i \sum_{r=0}^{n/2-1} w_{n/2}^{ir} x_{2r+1} \end{aligned}$$

Donde, los términos de la derecha son DFT de  $n/2$  puntos cada uno

$$u = F_{n/2} \begin{bmatrix} x_0 \\ x_2 \\ \vdots \\ x_{n-2} \end{bmatrix} \quad v = F_{n/2} \begin{bmatrix} x_1 \\ x_3 \\ \vdots \\ x_{n-1} \end{bmatrix}$$

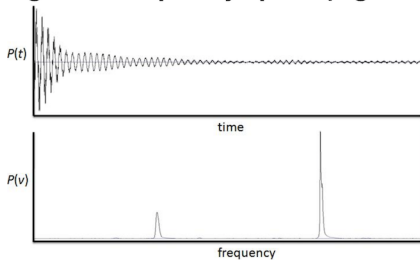
$$y_i = \begin{cases} u_i + \omega_n^i v_i & 0 \leq i < n/2 \\ u_{i-n/2} + \omega_n^i v_{i-n/2} & n/2 \leq i < n \end{cases} \quad \text{or} \quad y_{i+n/2} = u_i + \omega_n^{i+n/2} v_i$$

El resultado final se obtiene con operaciones de suma-multiplicación finales, en un tiempo  $T(n)=2T(n/2)+n=n\log(n)$

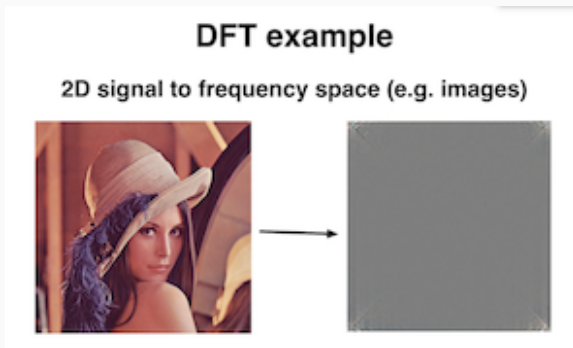
Ejemplo: **FFT** (Transformada rápida de Fourier):

## DFT example

1D signal to frequency space (e.g. sound)



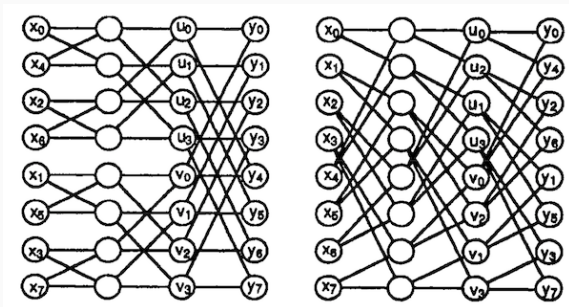
Ejemplo: **FFT** (Transformada rápida de Fourier):



# Paralelismo directo

Ejemplo: **FFT en paralelo:**

FFt de 8-puntos:



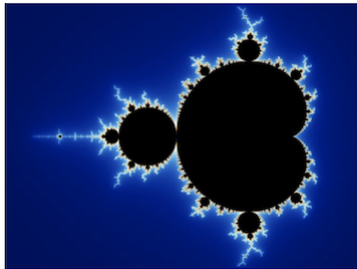
La distribución original puede ser reordenada como en la derecha, mejorando la complejidad a  $O(n)$ . El reordenamiento puede ser complejo para  $n$  grande.

Ejemplo: **Mandelbrot set**:

Es un set de

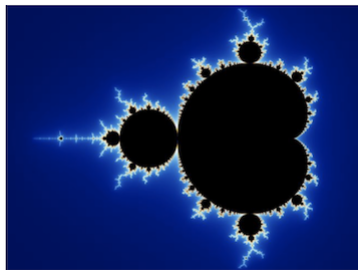
puntos complejos, que se calculan  
a través de la iteración de una  
función compleja no-divergente  
(iterada desde  $z=0$ ) de forma

$$z_{k+1} = z_k^2 + c$$



Ejemplo: **Mandelbrot set**:

donde  $z_{k+1}$  es la iteración  
(k+1) del número complejo  
 $z = a + bi$  (con  $i = \sqrt{-1}$ ), así  
como  $c$  es un número complejo.  
Se itera hasta que la longitud  
del vector  $z_{len} = \sqrt{z_{real}^2 + z_{imag}^2}$   
es mayor que 2





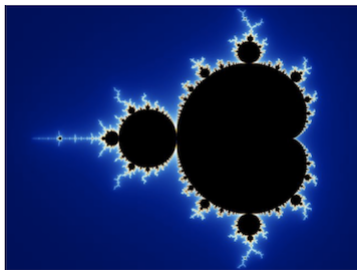
Ejemplo: **Mandelbrot set:**

**En paralelo:**

Cada pixel puede ser calculado independientemente del resto (el cálculo individual del pixel no es paralelizable)

**Distribución estática de tareas:**

Cada pixel necesita un número distinto de iteraciones, por lo que cada proceso recibe una porción de la imagen, y no terminan sincronizadas entre ellas, lo que torna ineficiente al algoritmo.

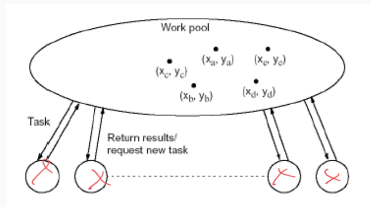


# Paralelismo directo

Ejemplo: **Mandelbrot set**:

**Distribución dinámica de tareas:** Las tareas en paralelo se distribuyen en un área común (work pool), desde donde cada proceso obtiene una nueva tarea en caso quede inactivo

Dados  $n$  pixel y un número máximo de iteraciones que depende de  $c$   $t_s \leq \max\_iter \times n$ , y por ello el tiempo es  $O(n)$



# Paralelismo directo

Ejemplo: **Mandelbrot set**:

Existen 3 pasos:

- Transferencia

de datos a  $p-1$  procesos,

$$t_{comm_1} = (p - 1)(t_{startup} + t_{data})$$

- Cálculo

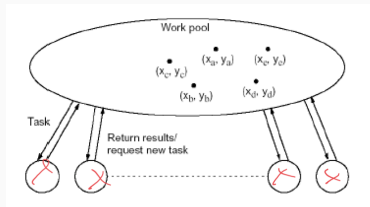
en paralelo  $\cdot t_{comp} \leq \frac{\text{max\_iter} \times n}{p-1}$

- Transferencia de resultados  
al master constante,  $t_{comm_2} = k$

En total

$$t_p = t_{comp} + t_{comm_1} + t_{comm_2}, \text{ ty la}$$

eficiencia se acerca a  $p$  si  $\text{max\_iter}$  es grande. El ratio  
cálculo/comunicación es  $O(n)$ , si  $p$  es constante.



## Uso de random en paralelo

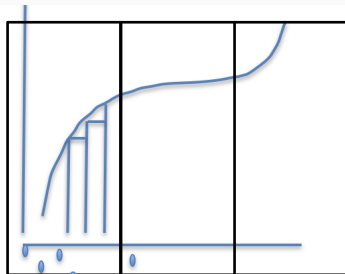
---

Utiliza principios de cálculo probabilístico y estadística, para aproximar soluciones de problemas complejos. Por ello, los resultados son correctos con cierta probabilidad. Se utilizan:

- Confiabilidad de sistemas técnicos
- Operatividad (almacén, transporte)
- Estudio de sismos o fenómenos naturales - Finanzas

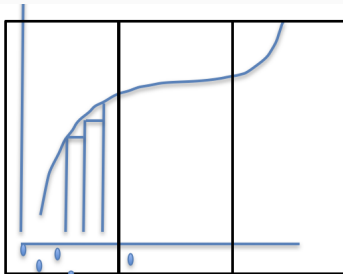
# Método Montecarlo

Dada una  
integral  $I = \int_{x_1}^{x_2} (x^2 - 3x) dx$ , ésta  
puede calcularse a través de la  
generación de  $N$  valores aleatorios  
entre  $x_1$  y  $x_2$ , y ser aproximados  
con  $I_{MC} = \sum_N (x^2 - 3x)$   
Ya que las iteraciones para suma  
o integrales son independientes, se  
paralelizan fácilmente, asignando  
a cada proceso un conjunto  
de números random entre  $x_1$  y  $x_2$



# Método Montecarlo

Estas se ejecutarán en cada proceso hasta que el criterio de aproximación se cumpla. Luego se recopila y suma la información en el nodo maestro. La generación aleatoria puede delegarse a cada proceso, evitando así tiempos de comunicación.

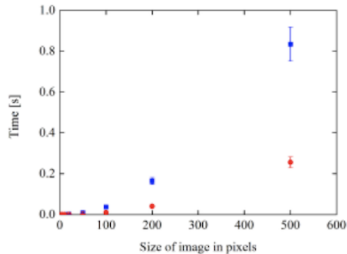


# Método Montecarlo

```
# pragma omp parallel \  
shared ( b, count, count_max, g, r, x_max, x_min, y_max, y_min ) \  
private ( i, j, k, x, x1, x2, y, y1, y2 )  
{  
# pragma omp for  
for ( i = 0; i < m; i++ ){  
    y = ( ( double ) (      i - 1 ) * y_max  
        + ( double ) ( m - i      ) * y_min )  
        / ( double ) ( m      - 1 );  
    for ( j = 0; j < n; j++ ){  
        x = ( ( double ) (      j - 1 ) * x_max  
            + ( double ) ( n - j      ) * x_min )  
            / ( double ) ( n      - 1 );  
  
        count[i][j] = 0;  
        x1 = x;  
        y1 = y;  
  
        for ( k = 1; k <= count_max; k++ ){  
            x2 = x1 * x1 - y1 * y1 + x;  
            y2 = 2 * x1 * y1 + y;  
  
            if ( x2 < -2.0 || 2.0 < x2 || y2 < -2.0 || 2.0 < y2 ){  
                count[i][j] = k;  
                break;  
            }  
            x1 = x2;  
            y1 = y2;  
        }  
    }  
}
```



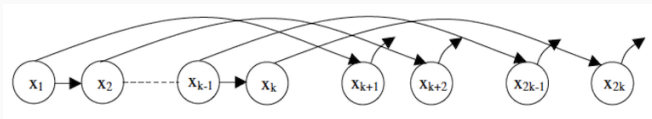
El tiempo de ejecución en paralelo es menor que el tiempo secuencial



# Generación random en paralelo

El generador 'linear-congruente' produce números pseudo-aleatorios, utilizando la función  $x_{i+1} = (ax_i + c) \bmod(m)$

Con a,c y m constantes, e.g. A=16807, m=2<sup>31</sup>-1, c=0



Se generan los primeros k valores en k procesos, que se utilizan para los siguientes k valores. Se logra una simplificación al ser m una potencia de 2

## **Sampling:**

Seleccionar una muestra representativa de un set de elementos y luego aplicarla al set original. Se usa en geometría, grafos, y algoritmos de string matching.

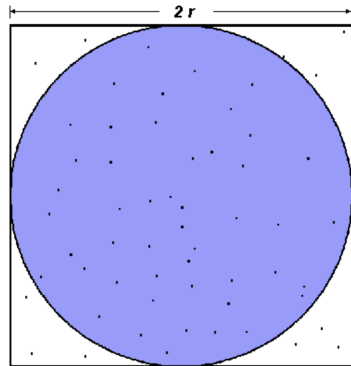
## **Load balancing:**

Una forma de particionar un número grande de elementos en una colección más homogénea, es asignando aleatoriamente cada elemento a un subgrupo.

**Symmetry breaking:** Por ejemplo, seleccionando un número elevado de vértices independientes en un grafo. Si uno decide unirse al set, el resto no deberá unirse. Es difícil de paralelizar, si la estructura de cada vértice es similar. El problema se resuelve usando randomización para romper la simetría entre los vértices.

# Método de Monte Carlo para calcular PI

- Genera  $N$  puntos random en el cuadrado exterior
- Determina el número de puntos  $n$  dentro del círculo
- Siendo el área del círculo  $\pi * r^2$ , y el área del cuadrado  $4 * r^2$
- Entonces  $n/N = \pi/4$ , o  $\pi = 4 * n/N$
- Generando un número suficiente de puntos se logra un resultado mas exacto
- $10^3 < N < 10^6$



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

# Secuencia Fibonacci

¿Qué tan rápido se reproducen conejos en condiciones ideales? (Fibonacci, 1202)

Conejos se reproducen en parejas cada mes y nunca mueren

Los números

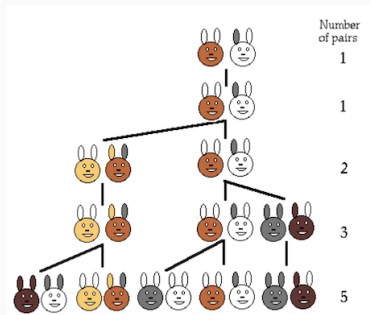
Fibonacci son la secuencia:

0,1,1,2,3,5,8,13,21,34,55,89,144,....

Cada número es la suma de los previos dos números, o

$$F_n = F_{n-1} + F_{n-2},$$

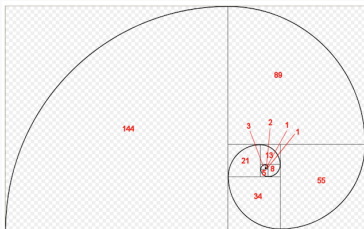
con  $F_0 = 0$ ,  $F_1 = 1$



# Secuencia Fibonacci

Se utiliza un algoritmo recursivo

```
funcion fib(n){  
  if(n<2)  
    return n;  
  return fib(n-1)+fib(n-2);  
}
```








## Dynamic multithreading:

Aquí, **spawn** paraleliza  $\text{Fibonacci}(n-1)$ , **sync** sincroniza para garantizar se usen los valores correctos de  $x, y$

```
Fibonacci(n):  
  if  $n < 2$ : | thread A  
    . return  $n$  | thread A  
   $x = \text{spawn Fibonacci}(n-1)$  | thread A  
   $y = \text{spawn Fibonacci}(n-2)$  | thread B  
  sync | thread C  
  return  $x+y$  | thread C
```



-  David B. Kirk and Wen-mei W. Hwu *Programming Massively Parallel Processors: A Hands-on Approach*. 2nd. Morgan Kaufmann, 2013. isbn: 978-0-12-415992-1.
-  Norm Matloff. *Programming on Parallel Machines*. University of California, Davis, 2014.
-  Peter S. Pacheco. *An Introduction to Parallel Programming*. 1st. Morgan Kaufmann, 2011. isbn: 978-0-12-374260- 5.
-  Michael J. Quinn. *Parallel Programming in C with MPI and OpenMP*. 1st. McGraw-Hill Education Group, 2003. isbn: 0071232656.
-  Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Program- ming*. 1st. Addison-Wesley Professional, 2010. isbn: 0131387685, 9780131387683.