Computación Paralela: Arquitecturas y programación

Programación paralela

Asignatura: Arquitectura de Sistemas Paralelos

Profesor: Francisco Javier Gómez Arribas
Departamento de Tecnología Electrónica y de las Comunicaciones

Contenidos

Programación Paralela

- Modelo de un programa paralelo
- Diseño y Metodología de programación
  - Particionado/Descomposición
  - Coordinación
  - Aglomeración/Asignación
  - Proyección
- Ejemplos de Programación

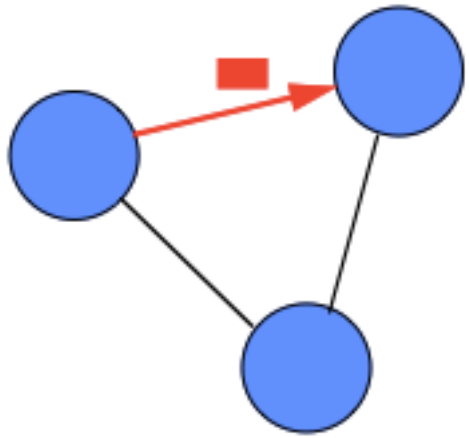
Un modelo de programación paralela sencillo

Un programa paralelo es un conjunto de tareas

Cada tarea que tiene datos locales se puede conectar con otras tareas por medio de canales.

Una Tarea puede:

- Realizar cálculos a partir de sus datos locales.
- Enviar/recibir datos de otras tareas.
- Crear nuevas tareas, o terminar por si misma.



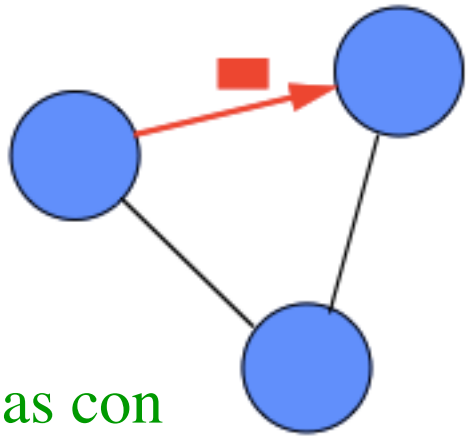
Una tarea que recibe queda bloqueada hasta que dispone de los datos

Características de un programas paralelos

Concurrencia: se incrementa al crear multiples tareas.

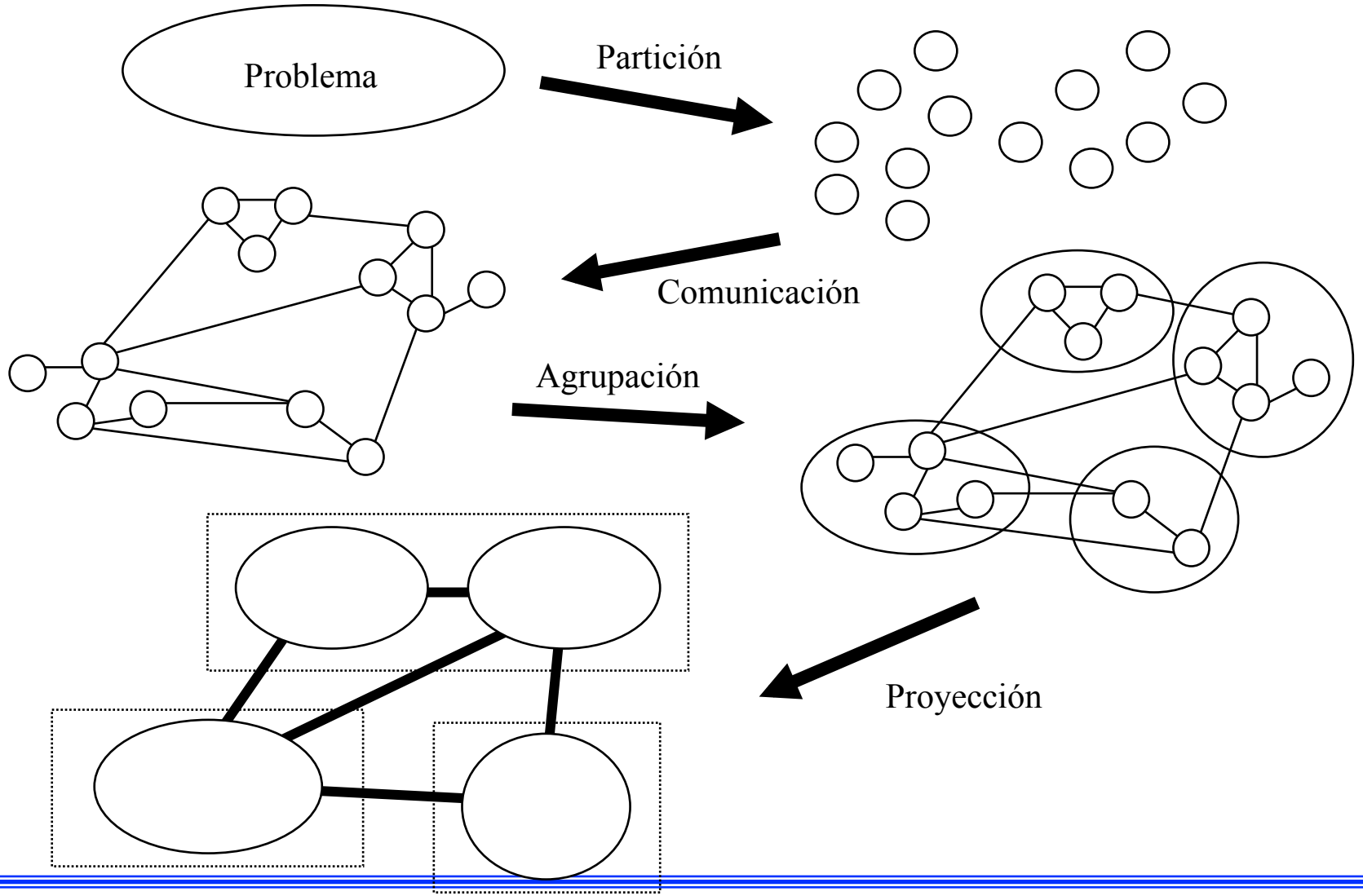
Escalabilidad: Mas tareas que nodos (elementos de proceso)

Localidad: Acceso local a datos si es posible



Diseño modular: Las unidades de diseño son las tareas con datos y subtareas locales. Se proyectan (mapping) en los nodos con el objetivo de mejorar el rendimiento.

Diseño de Algoritmos Paralelos



Metodología: Etapas de la Paralelización

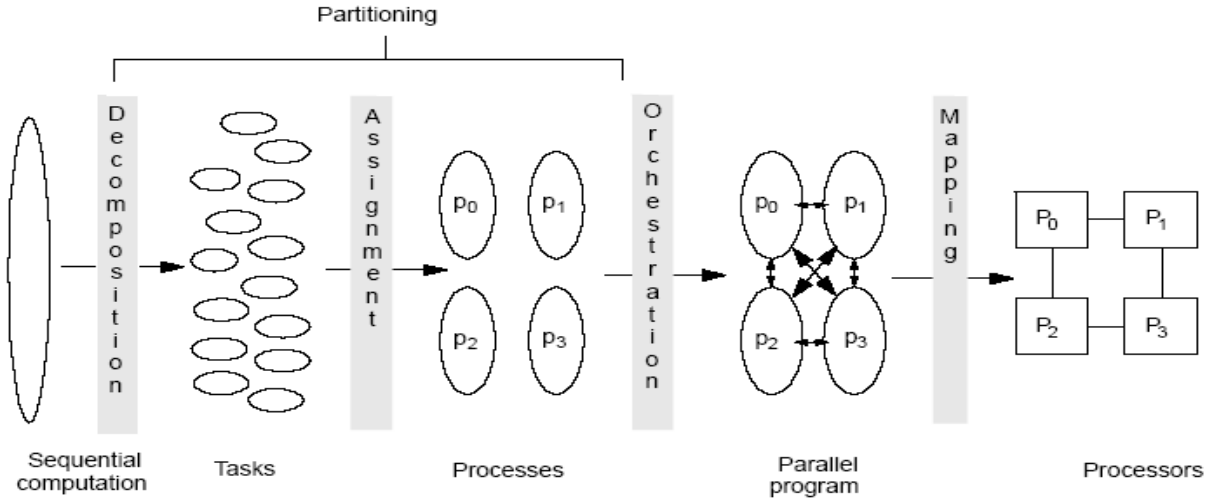
Partición/Descomposición: La computación se descompone en pequeñas tareas que son las unidades de concurrencia.

Coordinación/comunicación: Definir la denominación, comunicación y sincronización de las tareas.

Aglomeración/Agrupación: Las tareas se agrupan en procesos más grandes para optimizar el rendimiento y/o reducir costes de desarrollo.

Proyección: Los procesos se asignan a los procesadores maximizando la utilización y minimizando los costes de comunicación.

Metodología: Tareas y procesos



Se divide el cómputo en tareas las cuales a su vez serán repartidas entre procesos

TAREA

- Pieza arbitraria de trabajo no descompuesto en un cálculo paralelo
- Se ejecuta secuencialmente; la concurrencia existe solo entre tareas

⋆ **Las tareas pueden ser estáticas o dinámicas**

⋆ **El número de tareas disponibles puede variar con el tiempo**

PROCESO

- Entidad abstracta que realiza una o varias tareas asignadas
- Los procesos se comunican y sincronizan para realizar las tareas asignadas
- Los procesos aislan al programador del PROCESADOR

Es decir, la descomposición identifica concurrencia y decide el nivel en el cual se explotará

Metodología: Particionado

Objetivo: **Identificar las oportunidades de ejecución en paralelo**

- **Definir tareas (computación + datos)**
- **El número de tareas disponibles en un momento dado es una cota superior a la aceleración alcanzable**

¿Cómo particionar la Computación?

- Código Estructurado: Sugerido por el código (lazos, estruct de datos, ..)
- Código no Estructurado: Se puede recurrir a heurísticas dinámicas.

¿Qué ocurre con los datos?

- Pase de Mensajes: Se particionan con las computaciones
- Memoria Compartida: maximizar la localidad para mejorar el rendimiento

Estrategias de Particionamiento (datos vs operaciones del algoritmo)

- Descomposición en Dominios : Se particionan las estructuras de datos y se generan las tareas computacionales asociadas a cada partición.
- Descomposición Funcional : Se particiona las computacion en tareas funcionales y se asocian los datos maximizando la localidad.

Descomposición en dominios

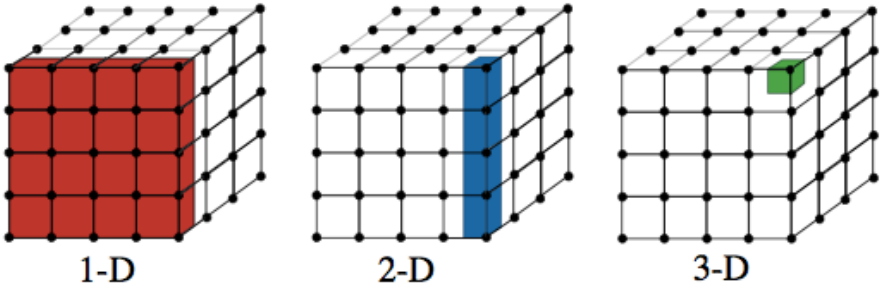
Particionado por descomposición en dominios

Método

1. **Los datos se particionan en grupos pequeños de tamaño similar**
2. **Se diseñan las tareas, a partir de cada uno de los grupos anteriores y todas sus computaciones asociadas.**
   - **Es el estilo usual en la programación SIMD.**

Elección de los Datos:

- **Es preferible la mayor estructura de datos o a la que se accede con mayor frecuencia**
- **En códigos grandes puede ser más eficiente varias descomposiciones diferentes.**

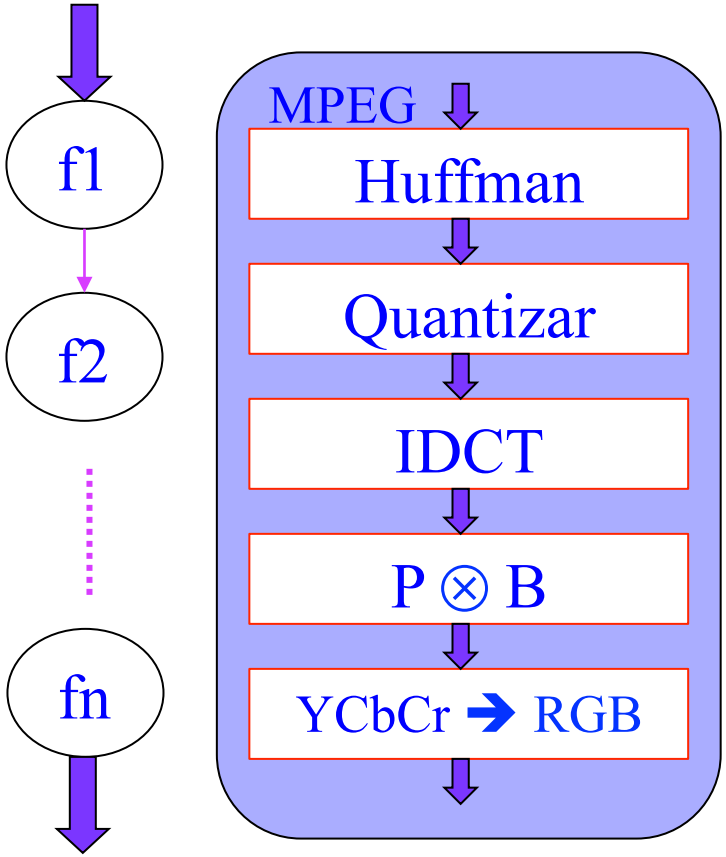


- 3-D decomp provides greatest flexibility

$$X_i = (X_{i-1} + 2*X_i + X_{i+1})/4$$

Particionamiento/Descomposición

Particionado por descomposición funcional



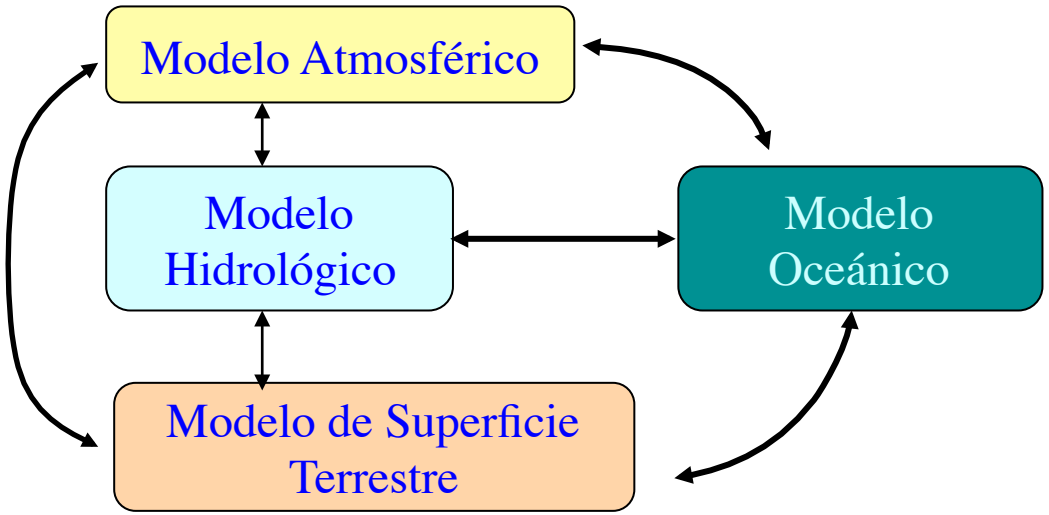
Mucho menos frecuente

Descomposición Funcional

Método:

1. **Las computaciones se descomponen en tareas disjuntas**
2. **Los datos se estructuran en función de las tareas diseñadas**

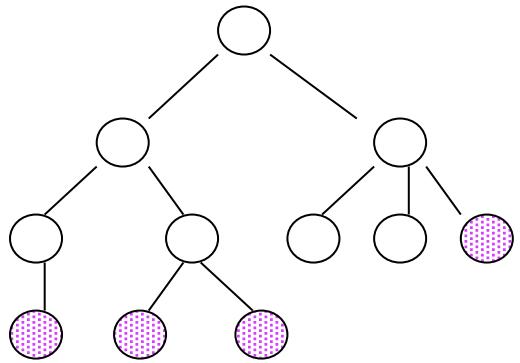
Modelo Climático



- En computación numérica, es más usual la descomposición en dominios.
- La descomposición funcional se suele combinar con descomposición en dominios en una jerarquía en dos niveles.

Ejemplo de Descomposición Funcional

```
Procedure search (A)
begin
    if (solution (A) ) then
        score = eval (A)
        report solution and score
    else
        foreach child A (i) of A
            search (A(i))
        endfor
     endif
end
```



Formulación recursiva de un algoritmo de búsqueda

Algoritmo

- Cuando se llama a la función para expandir la búsqueda en árbol primero se chequea si el nodo es una solución.
- Si no es solución se llama recursivamente a la misma función en cada uno de los nodos expandidos como descendientes.

Estructura de tareas

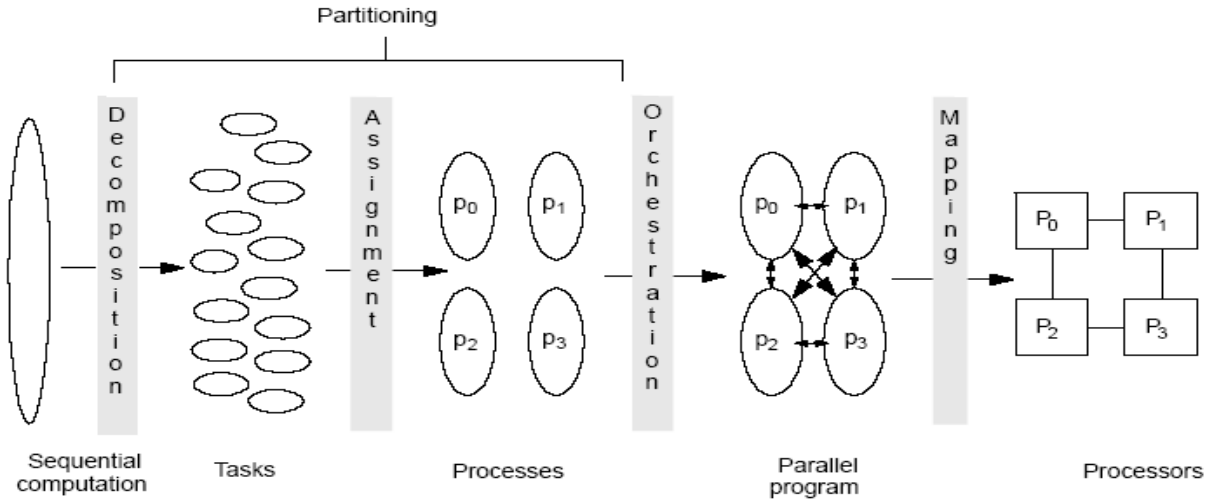
- Cada circulo representa un nodo en el arbol y por tanto una llamada a la función de búsqueda.
- Se crea una tarea por cada nodo conforme se explora el arbol.
- En un instante, algunas tareas están activas expandiendo el arbol (sombreadas en la fig) y otras han alcanzado un nodo solución, o esperan a que sus descendientes les comuniquen una solución.
- Las líneas representan los canales utilizados para comunicar la solución.

Checklist del particionado

En el particionado elegido

1. **¿ El número de tareas definidas es un orden de magnitud superior al número de procesadores en el sistema paralelo?**
2. **¿Elimina redundancia en computación y almacenamiento?**
3. **¿Son las tareas de tamaño comparable?**
4. **¿Escala el número de tareas con el tamaño del problema?**
5. **¿Se han identificado varias particiones alternativas?**

Comunicación / Coordinación



- **nombrar (identificar) datos**
- **Estructural la comunicación: intercambio de datos con otros procesos**
- **sincronización y planifcación de tareas.**

Objetivos

- **reducción de costos de comunicación y sincronización**
- **promover la localidad de referencias a datos**
- **planificación adecuada de tareas para evitar latencias**
- **reducción del trabajo adicional (Overhead) para gestionar el paralelismo**

La arquitectura, el modelo de programación y el lenguaje de programación juegan un papel importante

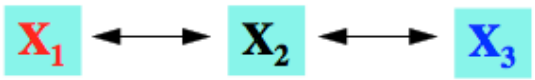
Comunicación / Coordinación

Se debe establecer el acceso a datos, y la comunicación y sincronización entre procesos.

Identificar los requisitos de Comunicación: Si se necesita los datos de otra tarea para la computación

Ejemplo : cálculo de diferencias finitas exige comunicación a primeros vecinos.

$$X_i = (X_{i-1} + 2*X_i + X_{i+1})/4$$



Partition creates one task per point

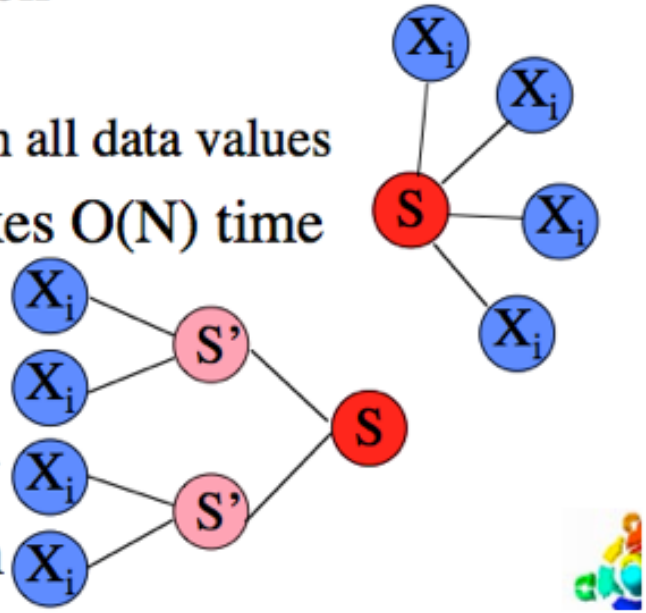
Alternativas: Comunicación/Sincronización:

- **Memoria compartida: Los datos se distribuyen a lo largo del espacio de memoria física, y la comunicación/sincronización se produce implícitamente a través de variables definidas en el espacio compartido.**
  - **La distribución de datos no es esencial en un modelo de memoria compartida, pero sí recomendable**
- **Paso de Mensajes: Los datos deben distribuirse explícitamente en las memorias locales y la comunicación/sincronización debe establecerse explícitamente** (las tareas deben saber cómo están distribuidos los datos)

Comunicación / Coordinación

Comunicaciones globales:

- Example: summation
  - S = sum $_{i=0:N-1}$ [X_i]
  - Value S depends on all data values
- Simple solution takes O(N) time
- Design principles
  - Distribute work
  - Allow concurrency
- O(log N) algorithm



Influencia de la Descomposición

- ***En Dominios*: La organización eficiente de la comunicación es usualmente compleja, debido a las dependencias entre las tareas.**
- ***Funcional*: La organización eficiente de la comunicación es inmediata, consecuencia directa de la descomposición y corresponde al flujo de datos entre tareas.**

Esquemas de Comunicación

Categorización de las estructuras de comunicación

- Comunicación Local vs Global
  - ***Comunicación Local*: Cada tarea se comunica con pocas tareas**
  - ***Comunicación Global*: Cada tarea se comunica con muchas (o todas las) tareas.**
- Comunicación Estructurada vs No estructurada
  - ***Comunicación Estructurada*: Una tarea y sus vecinos forman una estructura regular**
  - ***Comunicación No estructurada*: El patrón de comunicaciones es irregular.**
- Comunicación Estática vs Dinámica
  - ***Comunicación Estática*: La identidad de las tareas comunicantes no cambia con el tiempo.**
  - ***Comunicación Dinámica*: El patrón de comunicaciones cambia durante la ejecución del programa.**
- Comunicación Síncrona vs Asíncrona
  - ***Comunicación Síncrona*: Los productores y consumidores operan de forma coordinada (cooperan)**
  - ***Comunicación Asíncrona*: El consumidor obtiene datos sin la cooperación del productor**

Checklist del diseño de comunicaciones

1. **¿Realizan todas las tareas aproximadamente el mismo número de comunicaciones?**

2. **¿Cada tarea se comunica sólo con un pequeño número de vecinos?**

3. **¿Pueden realizarse las operaciones de comunicación de forma simultánea?**

4. **¿ Los cálculos asociados a diferentes tareas, puede proceder concurrentemente?**

Aglomeración

Agrupación de Tareas en Procesos (tarea mas grande):

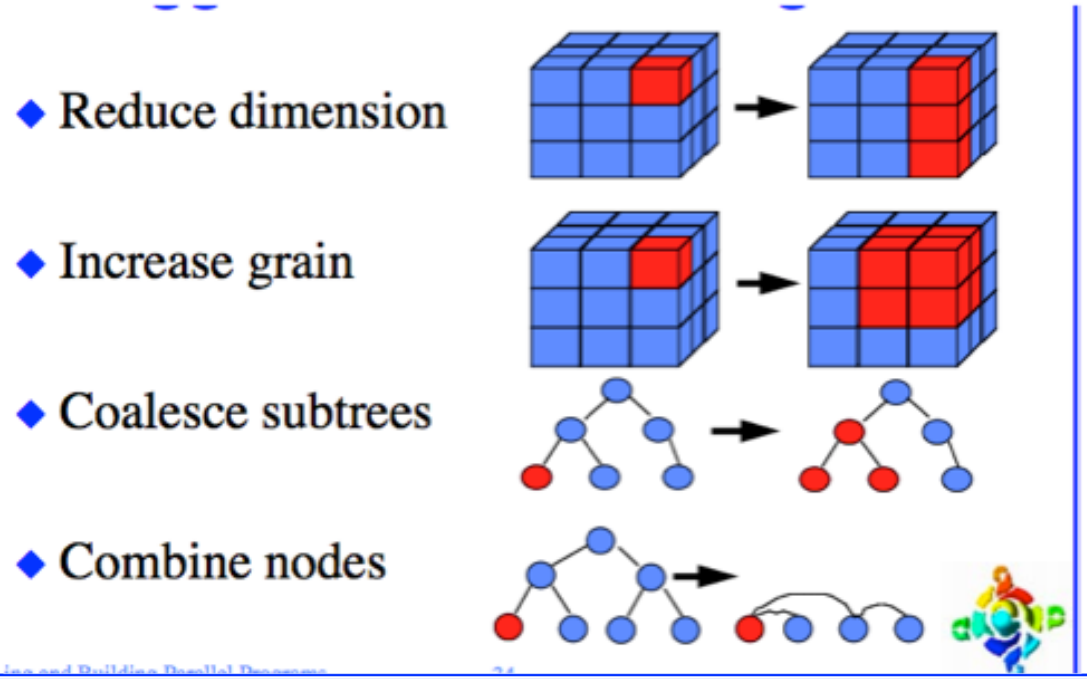
El número de procesos puede diferir del número de procesadores (multiprogramación)

- Interacción entre procesos pertenecientes a diferentes aplicaciones
- El número de procesos puede variar durante la ejecución de la aplicación
- La aglomeración especifica un mecanismo por el cual las tareas se asignan a procesos, o son seleccionadas por los procesos durante la ejecución

Aglomeración

Criterios que guían la Aglomeración:

- ***Aumento de la Granularidad*: Los costes de comunicación, sincronización y creación/gestión de procesos influyen críticamente en el rendimiento paralelo.**
  - **La granularidad puede controlarse en tiempo de compilación o de ejecución.**
- **Estrategias de aglomeración:**



Aglomeración

Criterios que guían la Aglomeración:

- ***Flexibilidad/Escalabilidad***
  - **No restringir innecesariamente la escalabilidad del algoritmo (la descomposición multidimensional es crítica)**
  - **La capacidad de crear un número variable de tareas es crítica para asegurar la portabilidad y escalabilidad (simultanear computación y comunicación)**
  - **Se requiere que haya más tareas que procesadores para asegurar una buena capacidad de balancear la carga**
- **Costes de Desarrollo**
  - **Diferentes estrategias de partición suelen tener costes de desarrollo distintos (seleccionar aquella que evite cambios extensos en el código)**
  - **Normalmente los algoritmos paralelos forman parte de grandes aplicaciones, por tanto hay que evaluar los costes de integración con los restantes módulos de la aplicación**

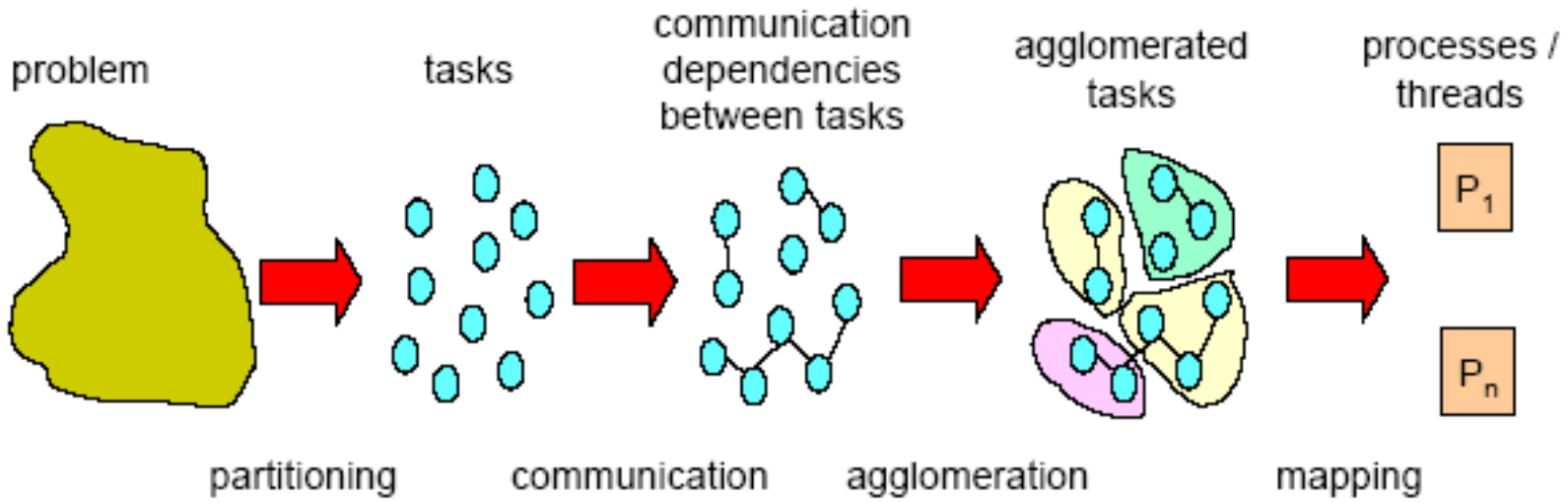
Checklist del diseño de la aglomeración

1. **¿Ha reducido la aglomeración los costes de comunicaciones aumentando la localidad?. En caso negativo se debe estudiar otra estrategia.**
2. **Si la aglomeración ha replicado computaciones (y datos), se debe verificar que esto es eficiente para un amplio abanico de problemas y plataformas.**
3. **Si se ha replicado los datos, hay que comprobar que ésto no restringe la escalabilidad de la aproximación**
4. **¿Son las tareas resultantes de similar costo en comunicaciones y computación?**
5. **¿El número de tareas agrupadas, todavía escala con el tamaño del problema?**
6. **Si la aglomeración ha eliminado oportunidades de ejecución simultánea, ¿ se ha verificado si todavía hay suficiente concurrencia para un rango amplio de plataformas?**
7. **¿Puede reducirse el número de tareas aún mas?**
8. **Si está paralelizando un código secuencial existente, ¿ se ha evaluado sus costes?. Si son altos, ¿se ha considerado otras estrategias totalmente diferentess**

Mapeo/Proyección

Cada proceso se asigna a un procesador de manera que se trata de maximizar la utilización de los procesadores y minimizar los costos de comunicación y sincronización

- La proyección es crítica en Máquinas Escalables



Aspectos de la Proyección de Procesos

- **¿Qué procesos deben ejecutarse en el mismo procesador? (multiprogramación,…)**
- **¿Qué procesos se ejecutan en un procesador concreto?**

Mapeo/Proyección

Estrategias de Proyección

- **Situar procesos concurrentes en procesadores diferentes (potenciar la concurrencia)**
- **Situar procesos que comunican frecuentemente en el mismo procesador (potenciar la localidad)**
- ***Descomposición Estructurada en Dominios*:**
  - **Proyección directa, minimizando comunicaciones.**
- ***Descomposición No-Estructurada Estática en Dominios*:**
  - **Heurística de proyección con algún esquema estático de balanceo de la carga.**
- ***Descomposición No-Estructurada Dinámica en Dominios*:**
  - **Heurística de proyección con algún esquema dinámico de balanceo de la carga**
- ***Descomposición Funcional*:**
  - **Heurística de proyección con algún esquema de planificación de tareas estático o dinámico.**

El problema de la Proyección es NP-Completo

Checklist del diseño de la proyección

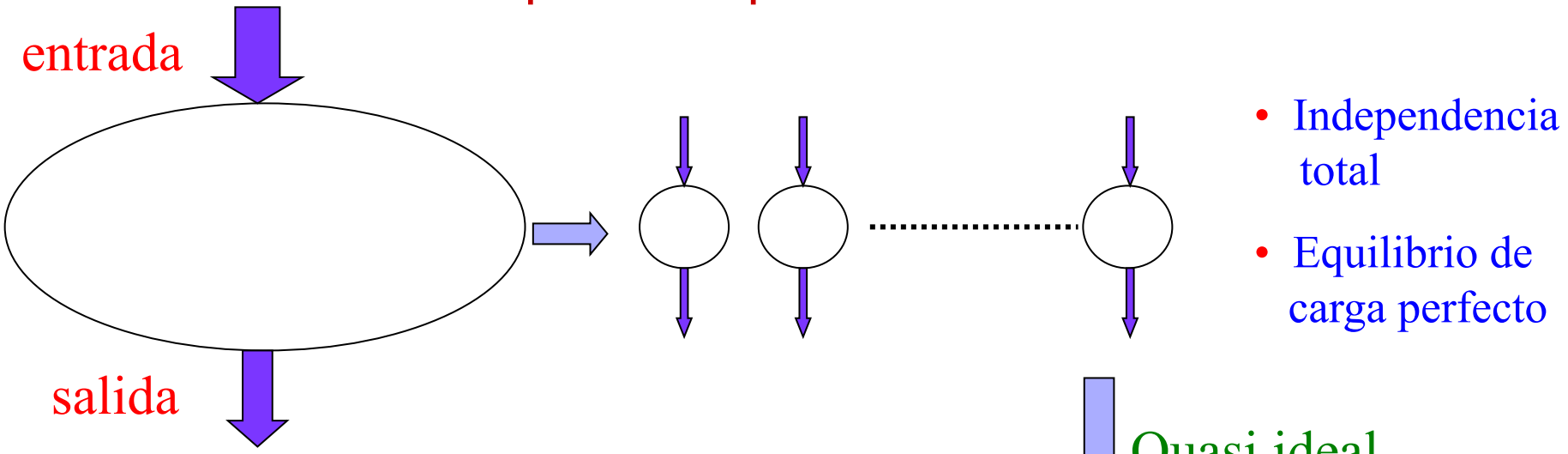
1. **Si se ha considerado un diseño SPMD y es muy complejo,**
   ¿Se ha considerado uno basado en la creación dinámica de tareas?

2. **Si se usa un esquema de balanceo de la carga centralizado,**
   ¿Se ha verificado que el master no es un cuello de botella?

3. **Si se usa un esquema dinámico de balanceo de la carga,**
   ¿se ha evaluado su costo?.
   Se debe intentar usar esquemas cíclicos o probabilísticos.

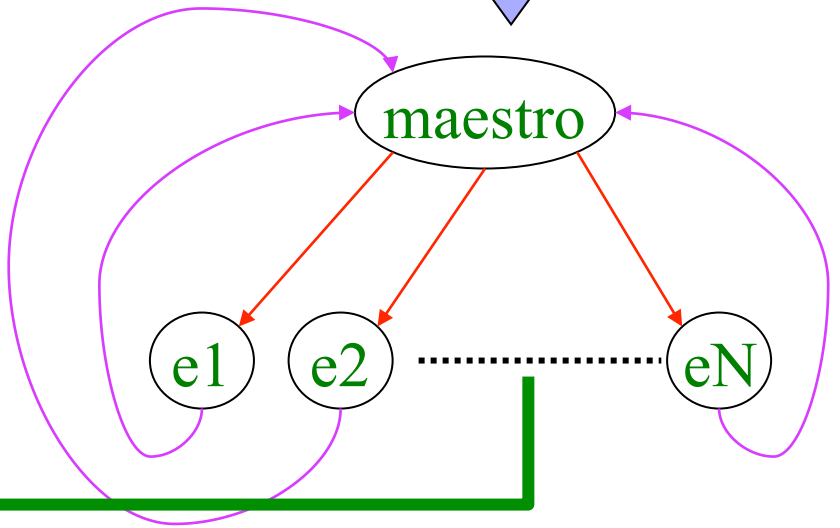
4. **Si se usa un esquema probabilístico,**
   ¿tiene un número razonable de tareas? (al menos 10 veces más que procesadores)

Computación paralela ideal



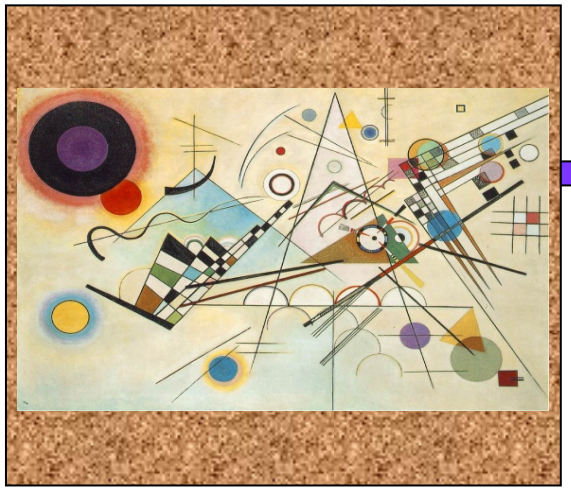
- Independencia total
- Equilibrio de carga perfecto

Quasi ideal



- Maestro reparte trabajos y recolecta resultados
- Los esclavos no se comunican entre sí
- Poca comunicación frente a computación

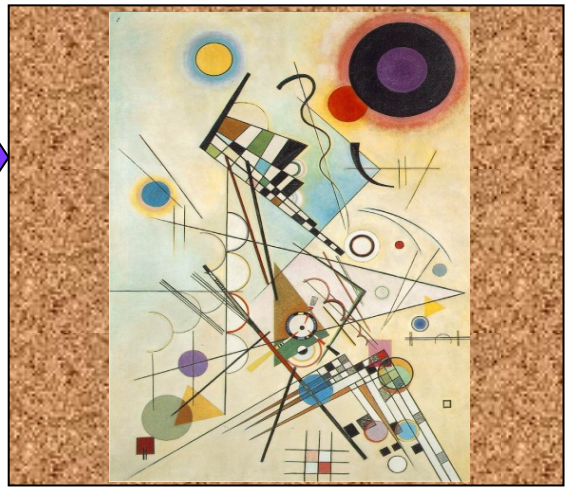
Paralelimo explícito :Transformación geométrica de imágenes



Girar:

$x' = x \cos\theta + y \operatorname{sen}\theta$

$y' = y \cos\theta - x \operatorname{sen}\theta$

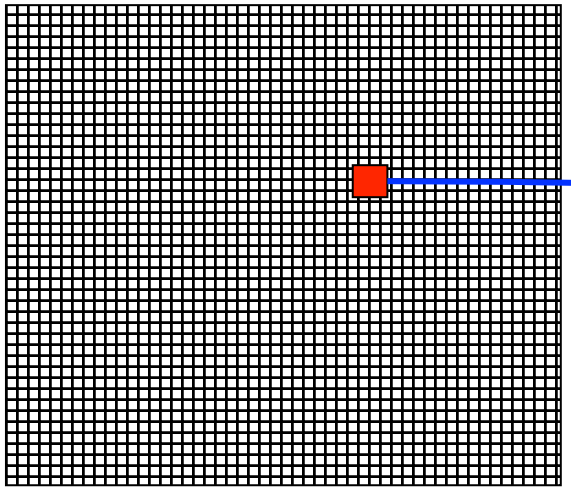


Desplazar, Escalar, ...

Igual para todo pixel

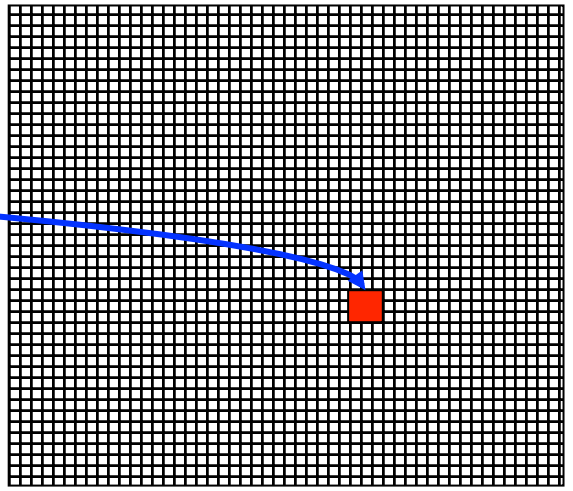
N

M



N

M



Paralelismo Compulsivo

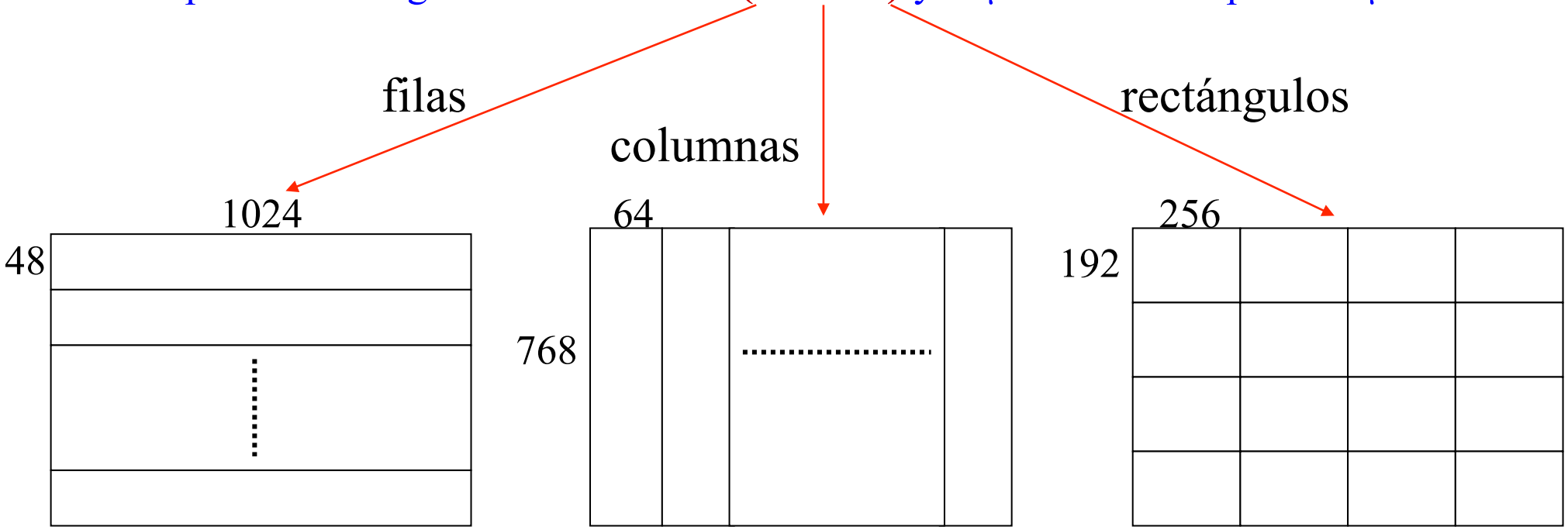
¿Un µP * pixel?

1024*768 = 786.432

Paralelimo explícito :Transformación geométrica de imágenes

Reparto de trabajo:

Si suponemos imagenes de 1024*768 **(786.432)** y 16μP => 49.152 pixels * μP



Rendimiento y eficiencia dependen:

- Sistemas multiprocesador vs multiprocesador.
- Memoria compartida y comunicaciones

Paralelimo explícito : Fractales-El conjunto de Mandelbrot

$$Z_{k+1} = Z_k^2 + Cj$$

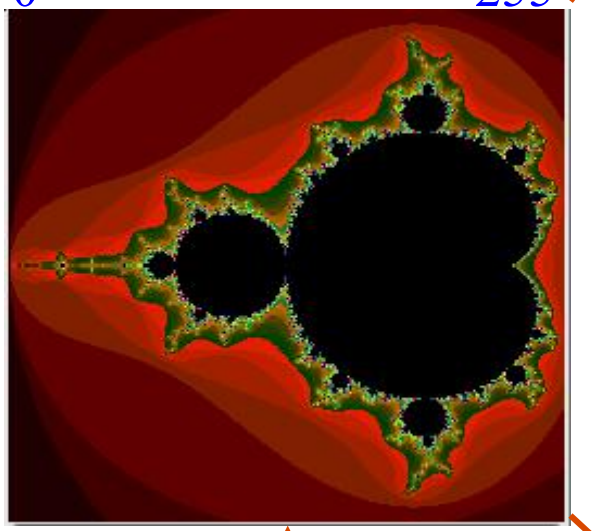
int colores[256][256]

1 …..... 2 ……. 3 …… 2 ……

1.. 2 .. 3…….. 4 --- 9 --- 256 0….. 462 ----- 3

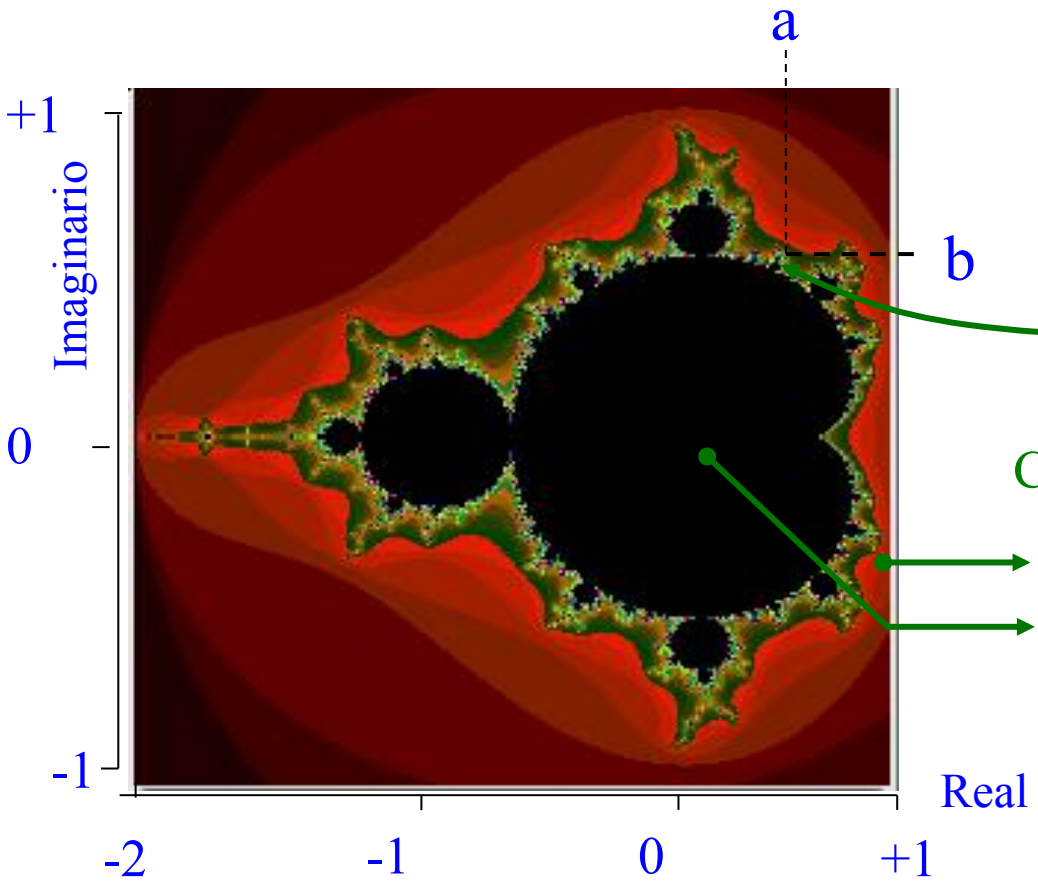
1 6 7 ---- 55 0 …… 20-5

1 ..... 2 ……... 3 ……... 2…….



256 colores

Paralelimo explícito : Fractales-El conjunto de Mandelbrot



$Z_{k+1} = Z_k^2 + Cj$

$Z_0 = 0$
$K = 0..\infty$

$Cj = a + b\,i$

$\exists m \;/\; |Zm| > 2 \Rightarrow Cj \notin \{M\}$

Condición de divergencia

$Cj \notin \{M\}$ diverge Zm => Color(m)

$Cj \in \{M\}$ no diverge => Negro

Conjunto de Mandelbrot {M}

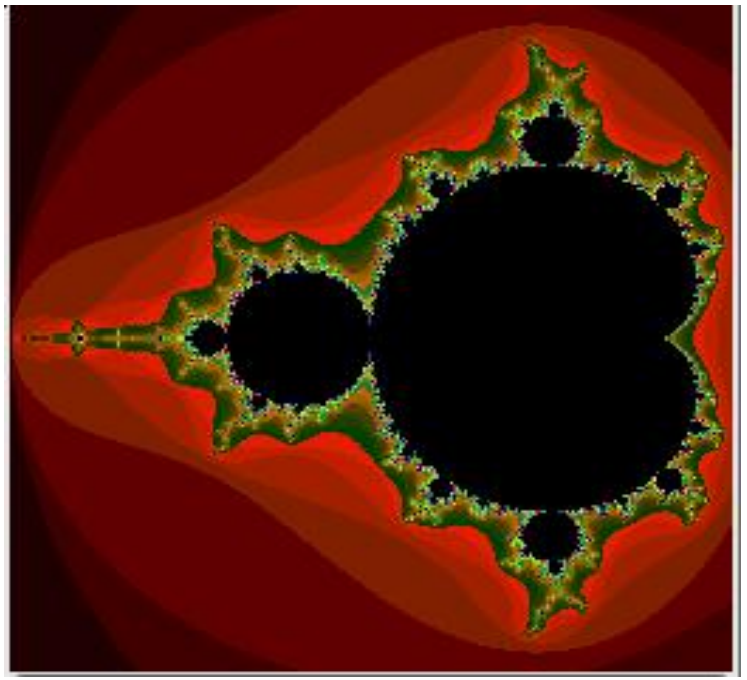
K = 0..N => # Colores a utilizar

Programa secuencial

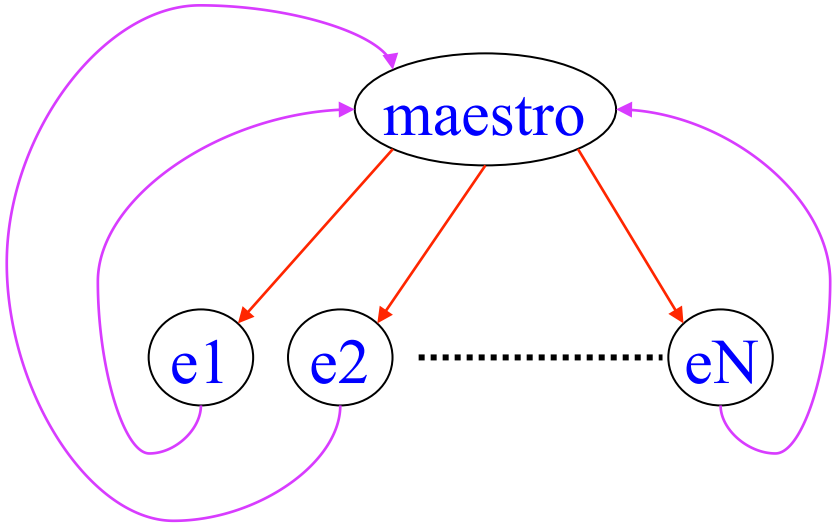
```
for (f=0; f<256; f++)
  for (c=0; c<256; c++) {
    pixelAPunto(f,c,&b,&a);
    color = mandelbrot(a,b);
    dibujarPixel(f,c,color);
  }
```

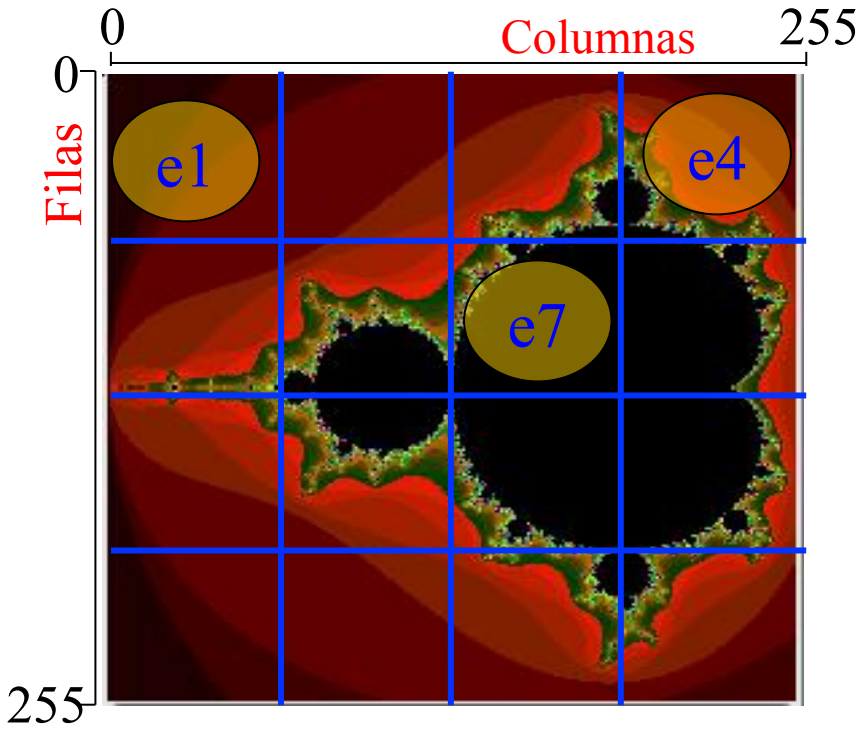
Paralelimo explícito : Fractales-El conjunto de Mandelbrot

```
#define MAX_ITER = 256

int mandelbrot (double A, double B) {
  double X = 0.0, Y = 0.0;
  double XX, YY, distancia;
  int i = 0;

  do {
       XX = X;
       YY = Y;
       X  = ((XX*XX) - (YY*YY)) + A;
       Y  = (2.0 * (XX*YY)) + B;
       i++;
       distancia = X*X + Y*Y;
  } while ((i < MAX_ITER) && (distancia <= 4.0));
  if (i == MAX_ITER) return 0;
  else               return i;
}
```



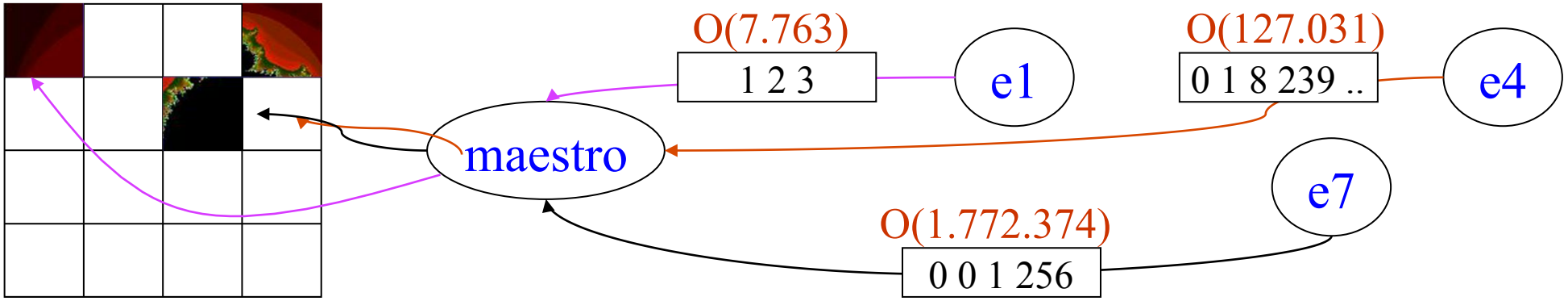
Paralelimo explícito : Fractales-El conjunto de Mandelbrot



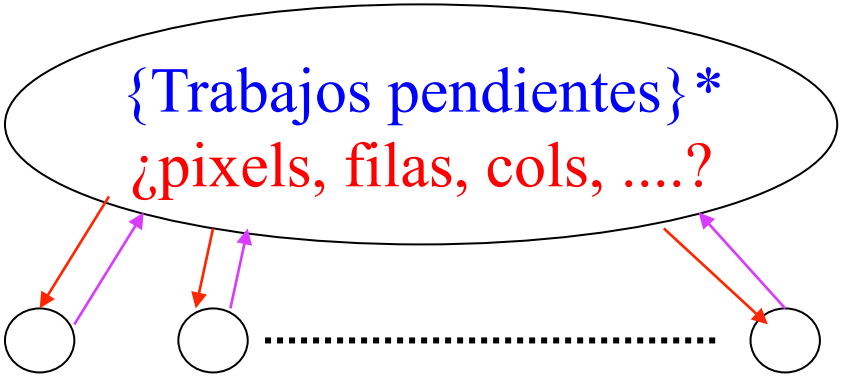


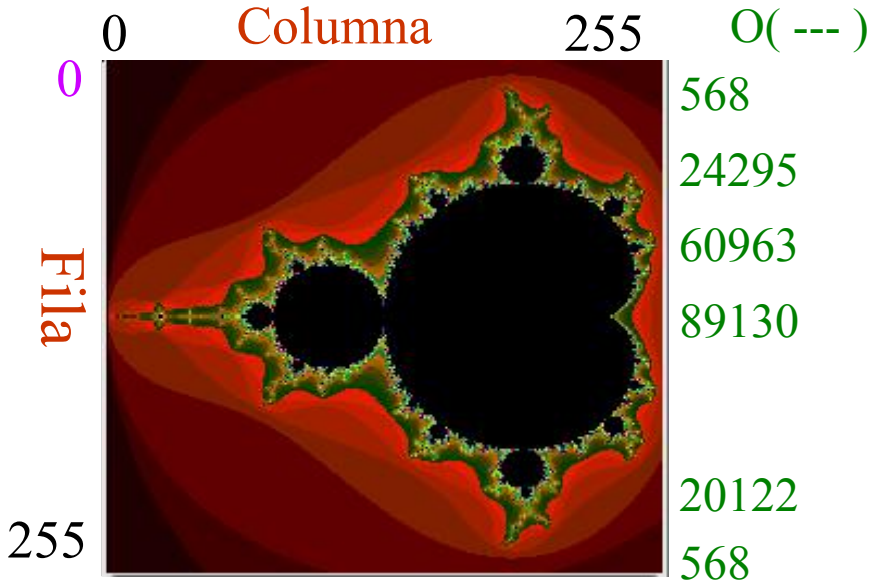
Asignación estática de trabajos
filas, columnas, cuadrantes

¡ No balanceado!



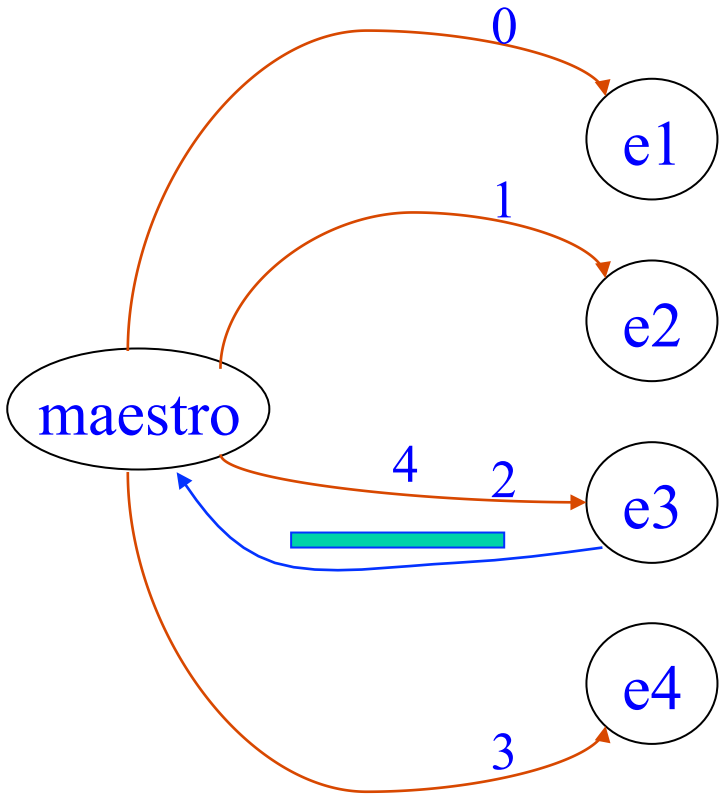
Paralelimo explícito : Fractales-El conjunto de Mandelbrot





b. Asignación dinámica de trabajos

Reparto bajo demanda



Paralelimo explícito : Los métodos de Monte Carlo

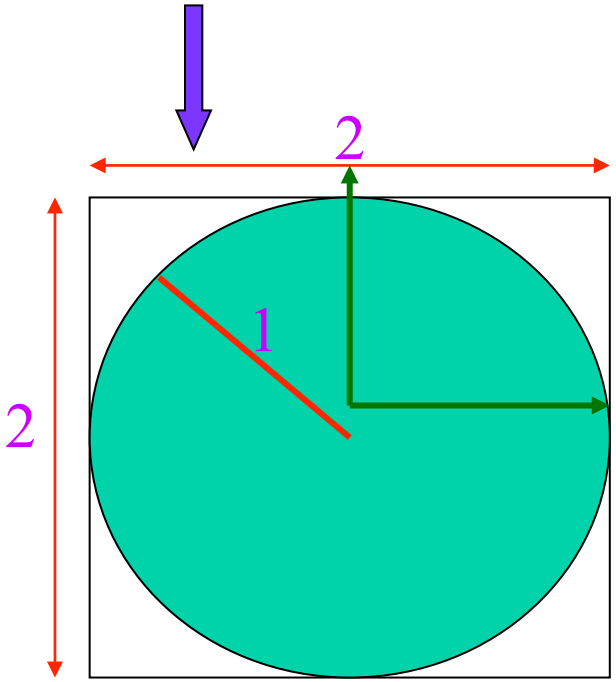
- **Idea:** Uso de números aleatorios – Casino de Monte Carlo

  - **Aplicaciones:**
    - Diseño de reactores nucleares
    - Cromo dinámica cuántica
    - Radioterapia contra el cáncer
    - Densidad y flujo de tráfico
    - Evolución estelar
    - Econometría
    - Pronóstico del índice de la bolsa
    - Prospecciones en explotaciones petrolíferas
    - Diseño de VLSI
    - Física de materiales
    - Ecología
    - Criptografía
    - Valoración de cartera de valores
    - Métodos cuantitativos de organización industrial

Los métodos de Monte Carlo

Se basan en la utilización de números aleatorios: Ejemplo1 => Π

Círculo de radio 1 inscrito en cuadrado de lado 2



¿M?

Suficientemente grande

Área del círculo = Π

Área del cuadrado = 4

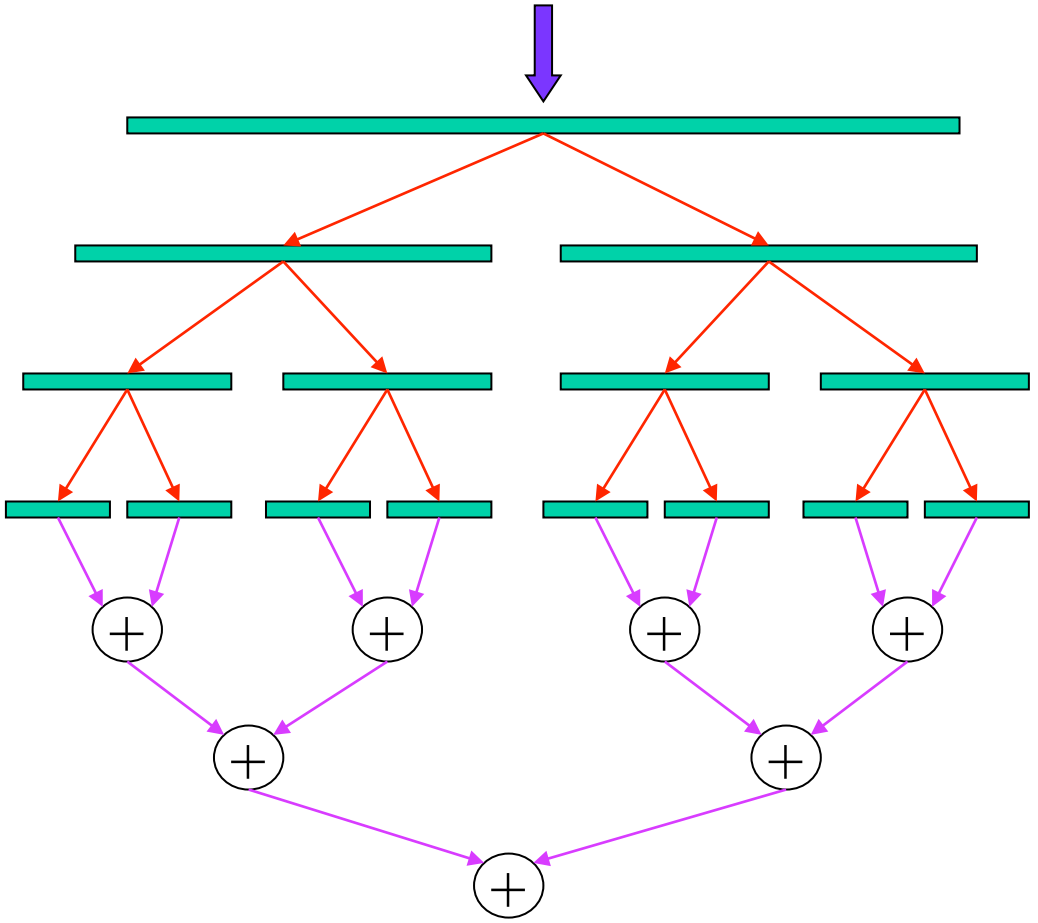
¿ Π / 4 ?

```
enCirculo = 0;
for (i=0; i<M; i++) {
    x = aleatorio(0.0, 1.0);
    y = aleatorio(0.0, 1.0);
    if ((x*x + y*y)<=1.0) enCirculo++;
}
PI = (4.0 * enCirculo) / (double) M);
```

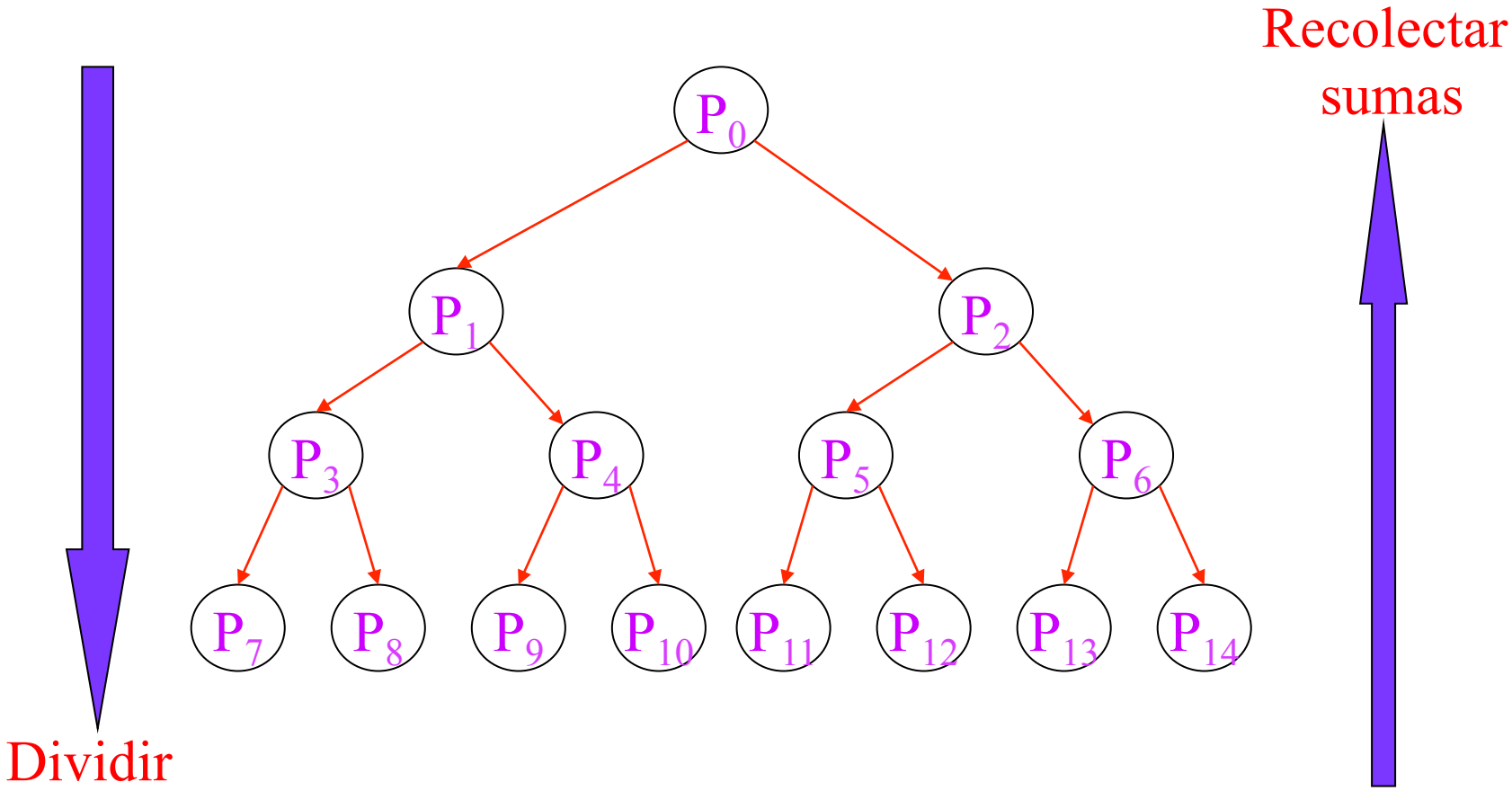
Divide y Vencerás

Divide y Vencerás: Particionar de forma recursiva, combinando resultados (Ej: QuickSort, SumarNúmeros)

```
int sumar(int *s) {
  if (card(s) <= 2)
    return (n1 + n2);
  else {
    dividir (s, s1, s2);
    suma1 = sumar(s1);
    suma2 = sumar(s2);
    return (suma1 + suma2);
  }
}
```

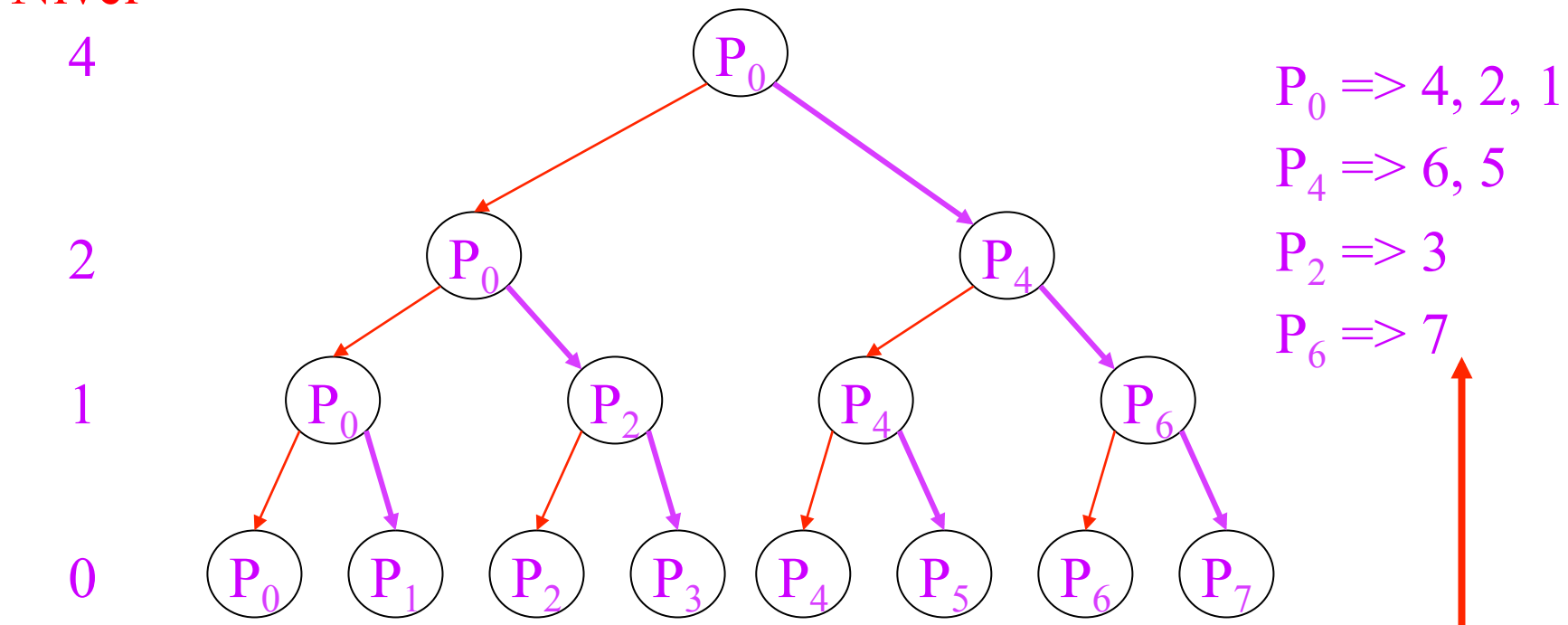


Divide y Vencerás (paralelización)



No trabajan todos los procesos todo el tiempo

Divide y Vencerás (paralelización)



¿ Cómo programarlo ?

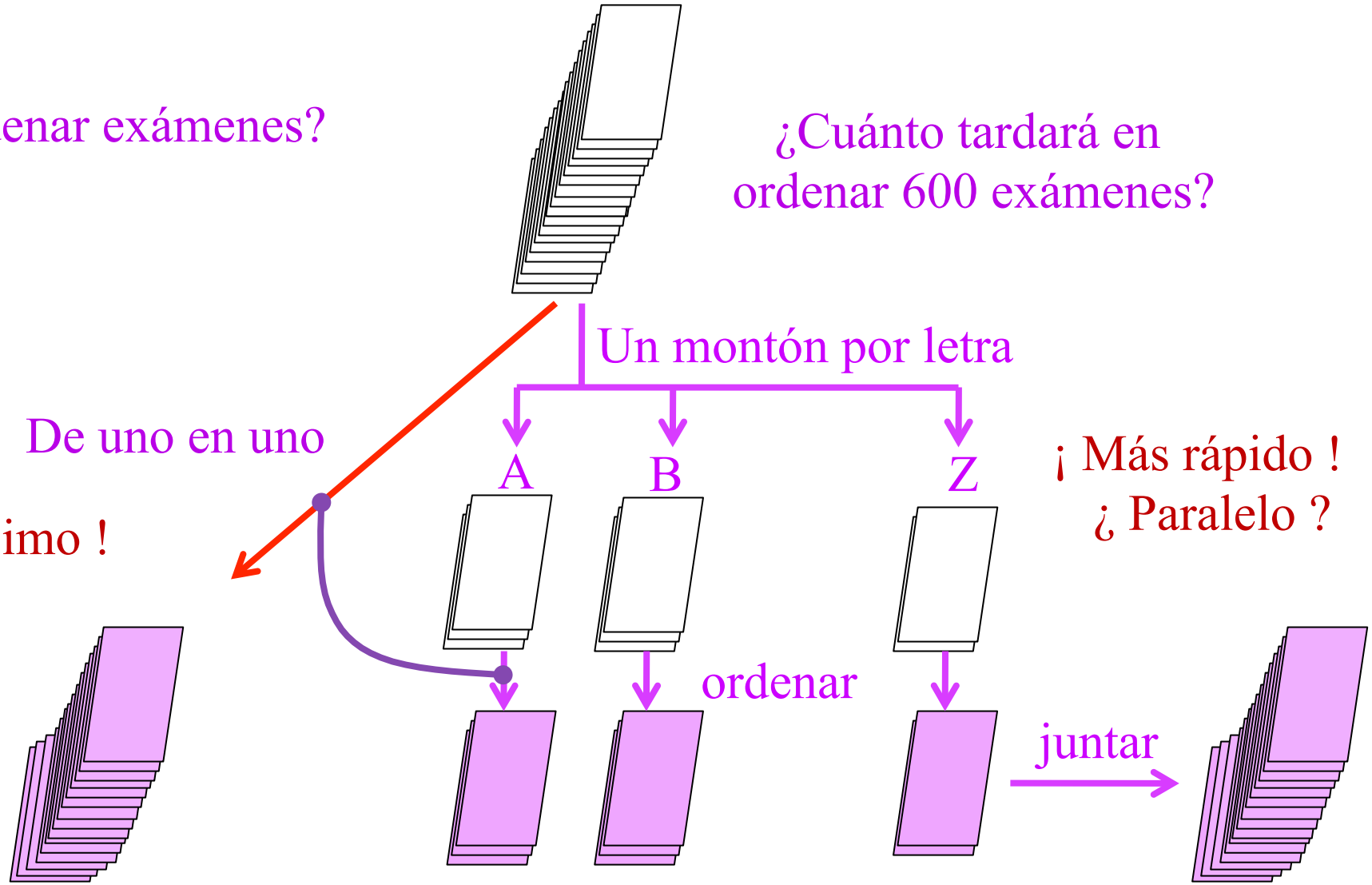
```
Recibir_Mi_Trozo
Distribuir_A_Descendientes
Procesar_Mi_Trozo
Recolectar_Sumas_De_Mis_Descendientes
Enviar_Mi_Suma_Total_A_Mi_Ancestro
```

→ ¿Cuáles?

Ordenación mediante cubetas

- ¿Ordenar exámenes?

¿Cuánto tardará en ordenar 600 exámenes?

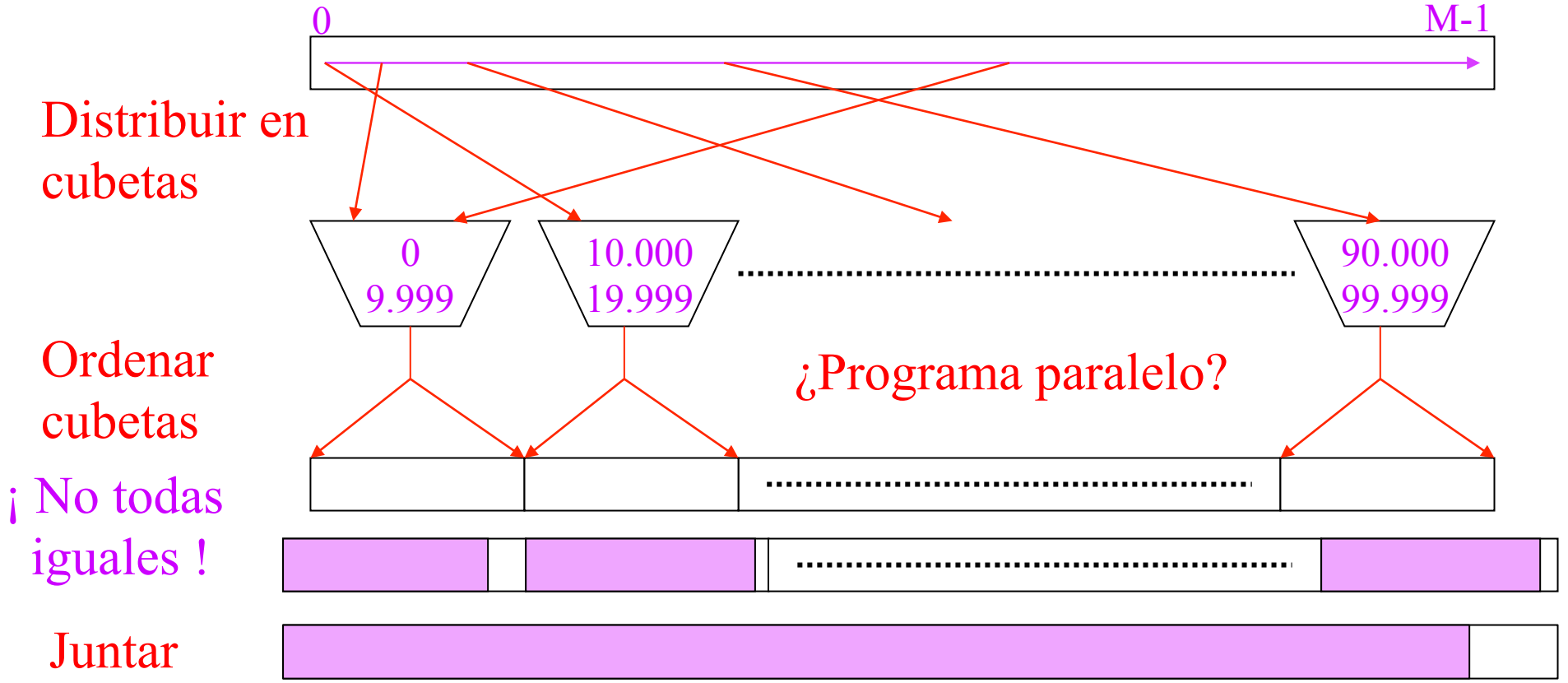


Ordenación mediante cubetas

M números (10.000.000) en un rango (sea: 0..99.999)

Uniformemente distribuidos en N intervalos regulares (sean: 10)

≈1.000.000 ∈ [0..9.999], ≈1.000.000 ∈ [10.000..19.999], ...



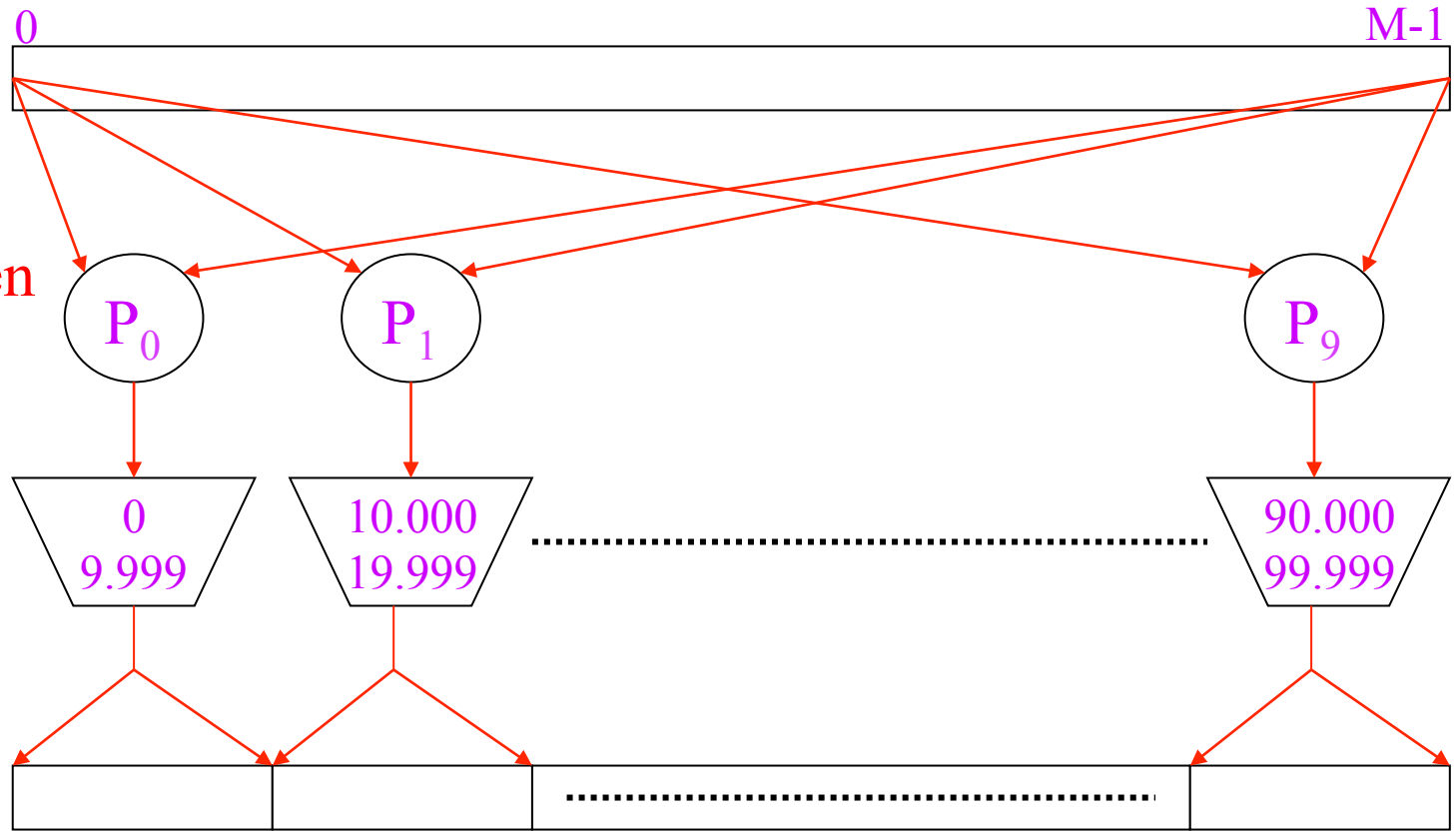
Ordenación mediante cubetas

Versión 1: Asociar una cubeta por proceso

Para todo Pi

Recolectar en mi cubeta

Ordenar mi cubeta



Todos los Pi necesitan el array completo

Ordenación mediante cubetas

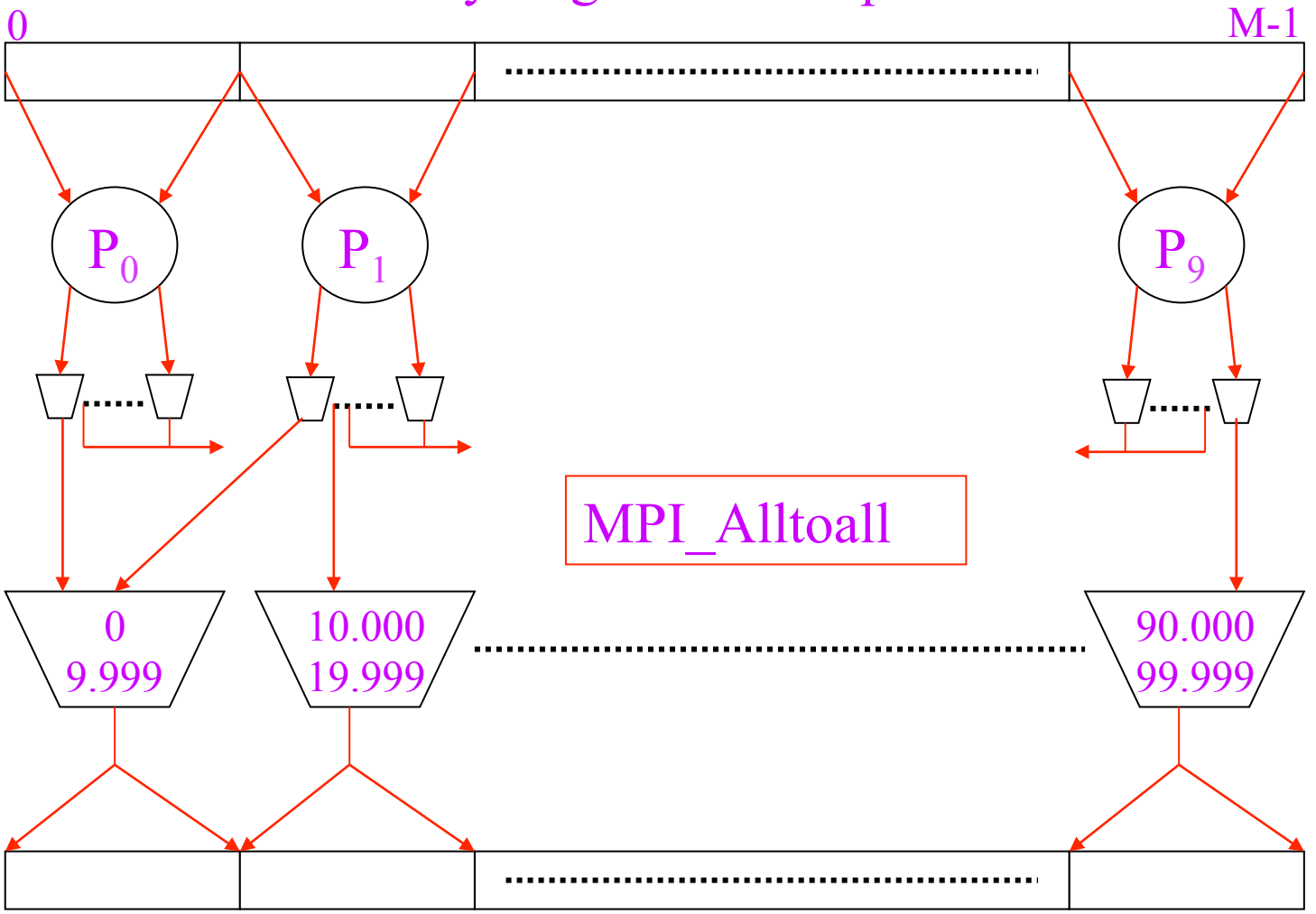
Versión 2: Asignar un trozo del array original a cada proceso

Para todo Pi

Recolectar en miniCubetas

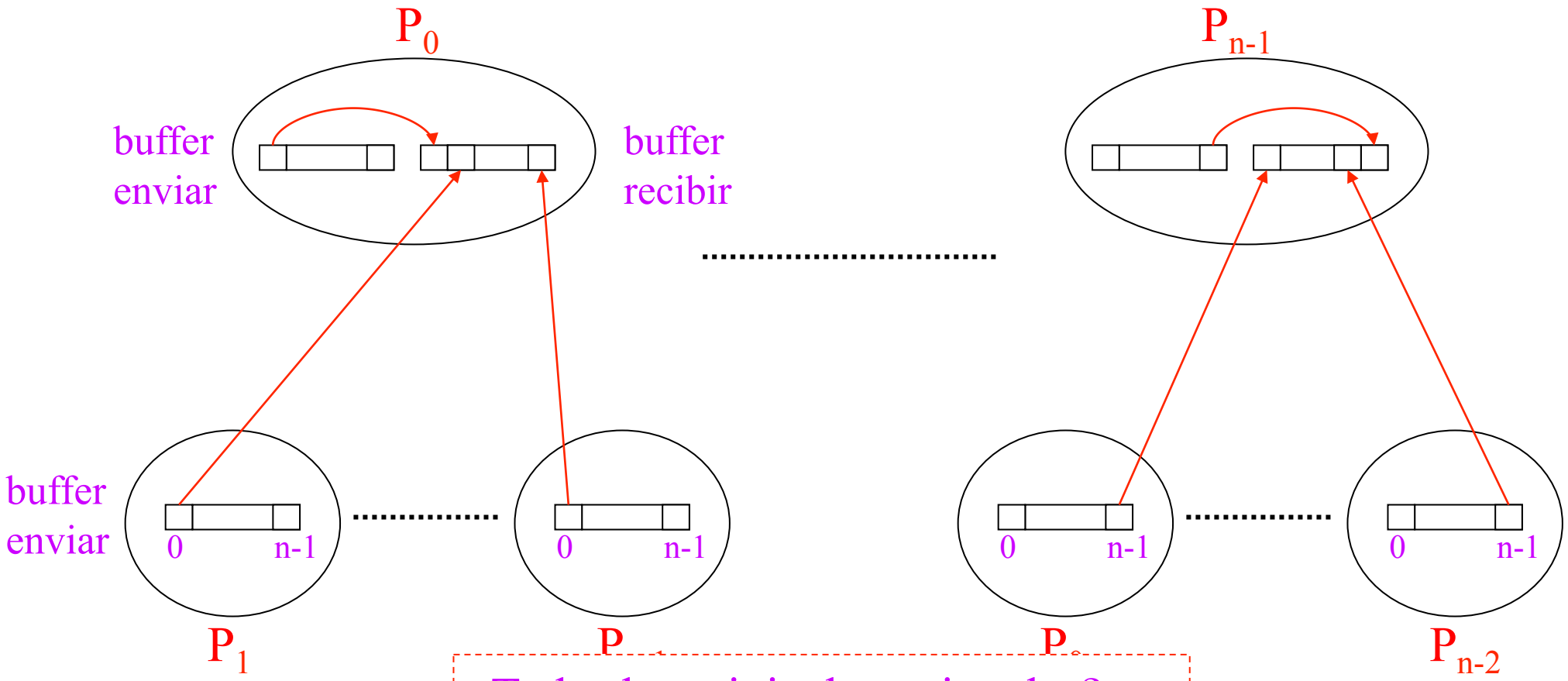
Env/Rec miniCubetas

Ordenar mi cubeta



Ordenación mediante cubetas

MPI_Alltoall (BufferEnvio, BufferRecepcion, MPI_Comm)



¿Todas las minicubetas iguales?

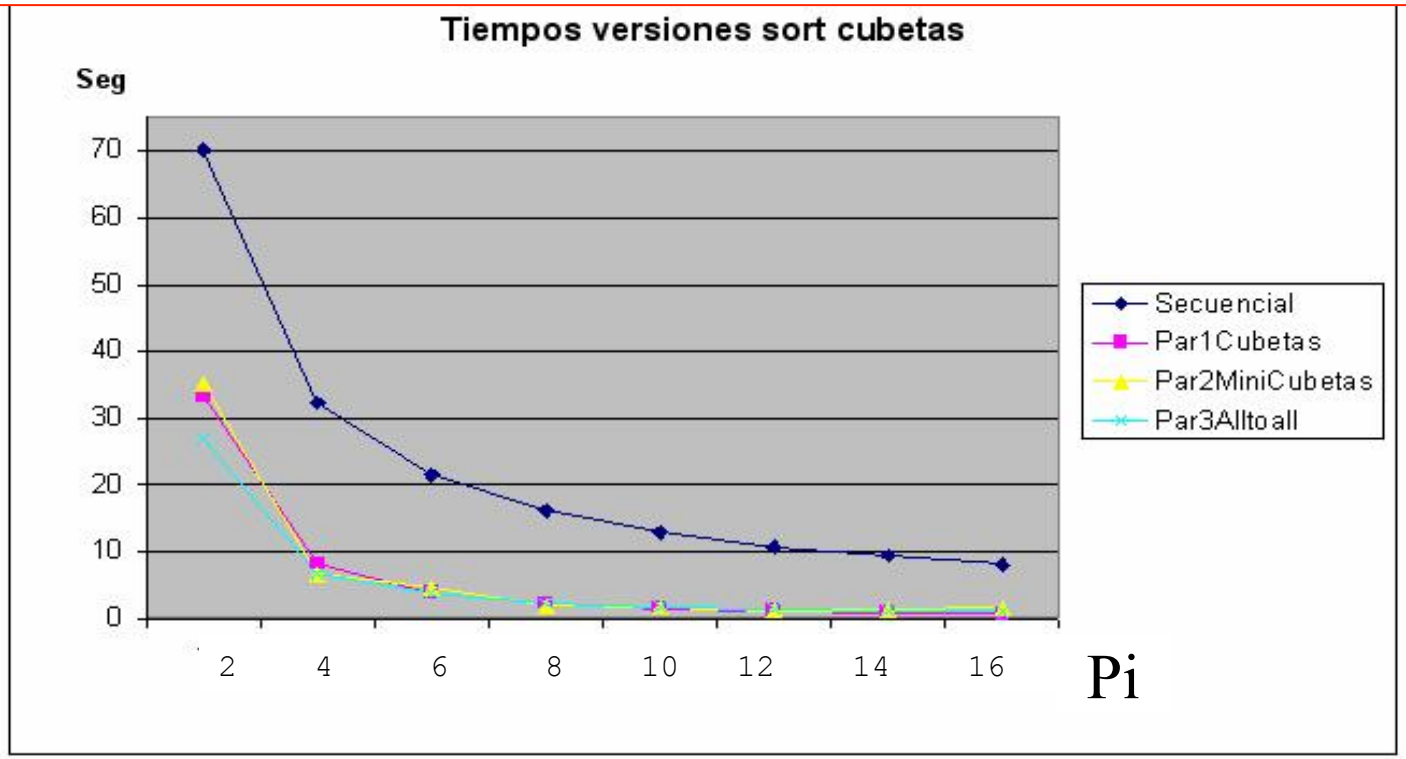
Tiempos ordenar Cubetas

NumCubetas	Sec	Par1	Par2	Par3
2	70,122	33,335	35,135	26,924
4	32,320	8,280	6,672	6,704
6	21,550	3,866	4,604	3,952
8	16,145	2,272	1,883	2,331
10	12,901	1,554	1,684	1,846
12	10,751	1,161	1,174	1,485
14	**9,421**	**0,961**	**1,382**	**1,247**
16	8,092	0,762	1,590	1,010

puro

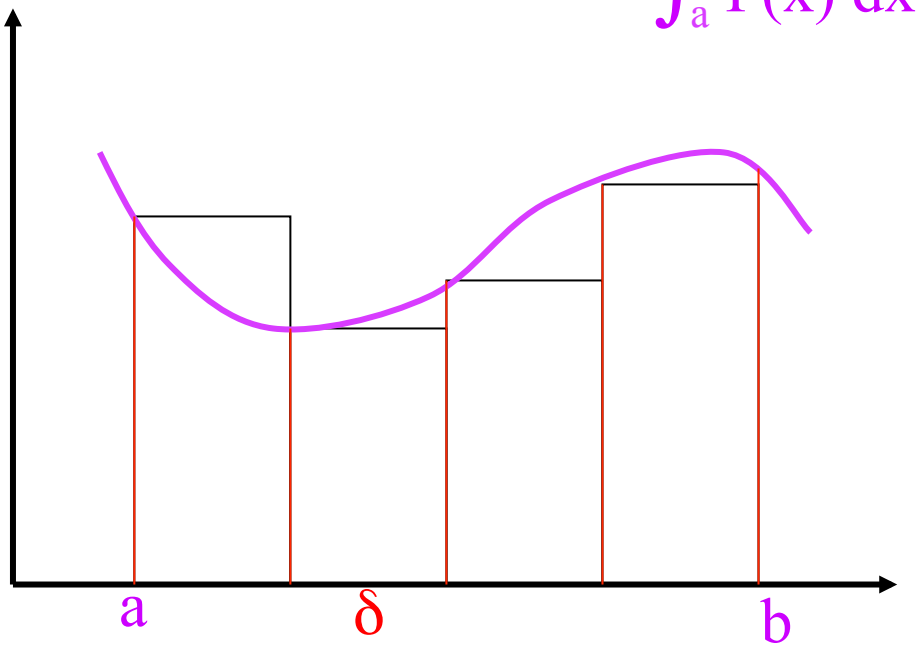
221,557

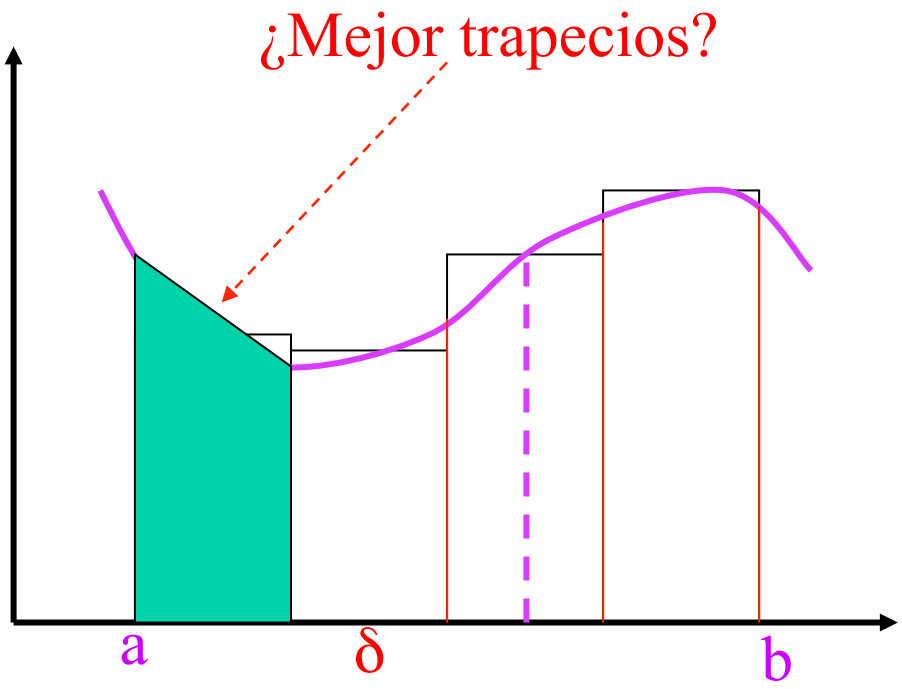
240.000 números, 2..16 cubetas/Pi y Selección Directa



Integración numérica

$$\int_a^b F(x)\,dx$$

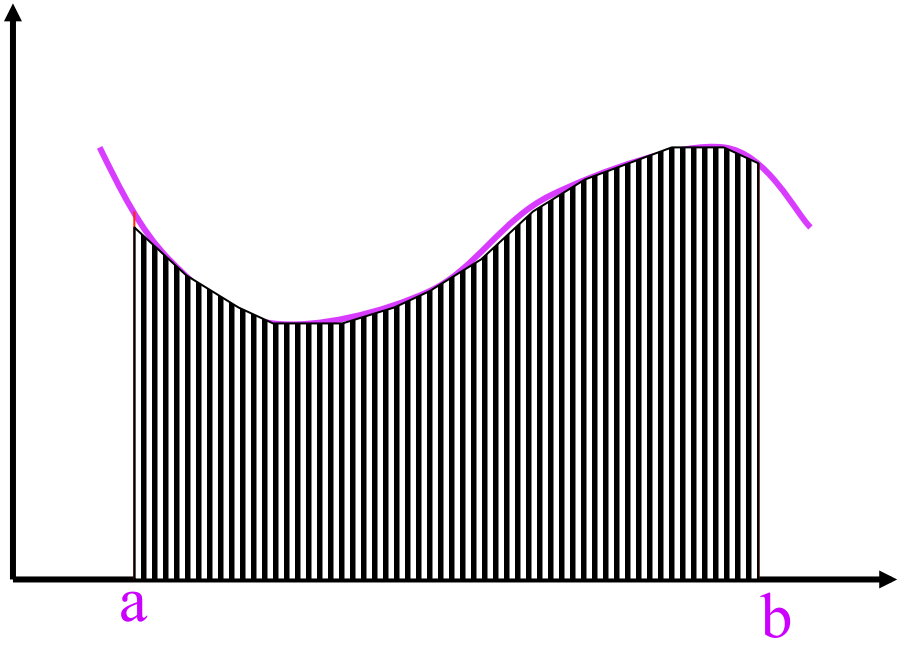




$\delta = (b - a) / N$ → Precisión

$$I = \delta \sum_{x=a}^{b-\delta} F(x)$$

Integración numérica



a) Particionamiento y Asignación Estática (4Pi)

$\delta = (b - a) / N$ (N=1.000.000)

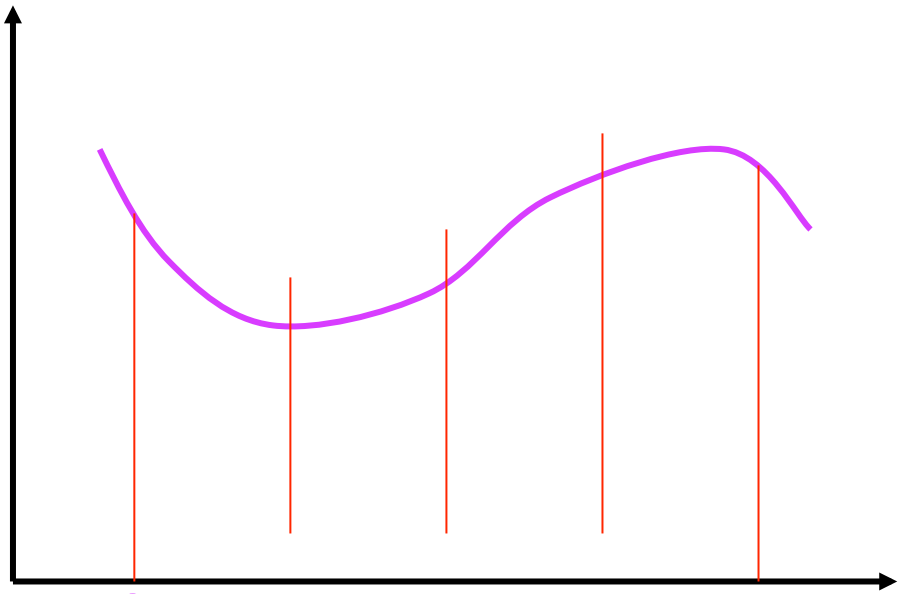
$\Delta = (b - a) / P$ (P=4)

Cada Pi calculará su trozo:

250.000 rectángulos (δ)

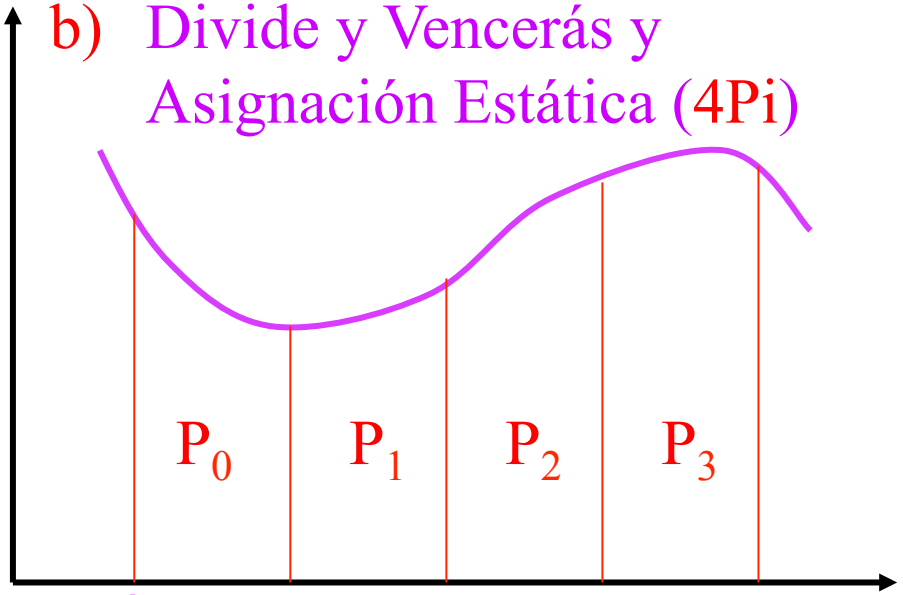
Conocimiento previo del N razonable

Integración numérica



$\delta = (b - a) / N \quad (N=¿?)$

b) Divide y Vencerás y Asignación Estática (4Pi)



N	Area
2	A_2
4	A_4
n	A_n
2n	A_{2n}

Reparto no equilibrado de trabajo

→ $|A_{2n}-A_n| < \text{cotaError}$