

2º curso / 2º cuatr.

Grado en
Ing. Informática

Arquitectura de Computadores

Tema 2

Programación paralela

Material elaborado por los profesores responsables de la asignatura:

Mancia Anguita – Julio Ortega

Licencia Creative Commons



ugr

Universidad
de Granada

ETSIIT

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación



ATC

Departamento de Arquitectura
y Tecnología de Computadores
UNIVERSIDAD DE GRANADA



Lecciones

- Lección 4. Herramientas, estilos y estructuras en programación paralela
- Lección 5. Proceso de paralelización
- Lección 6. Evaluación de prestaciones en procesamiento paralelo

Objetivos Lección 5

- Abordar la paralelización de una aplicación.
- Distinguir entre asignación estática y dinámica, ventajas e inconvenientes.

Bibliografía Lección 5

➤ Fundamental

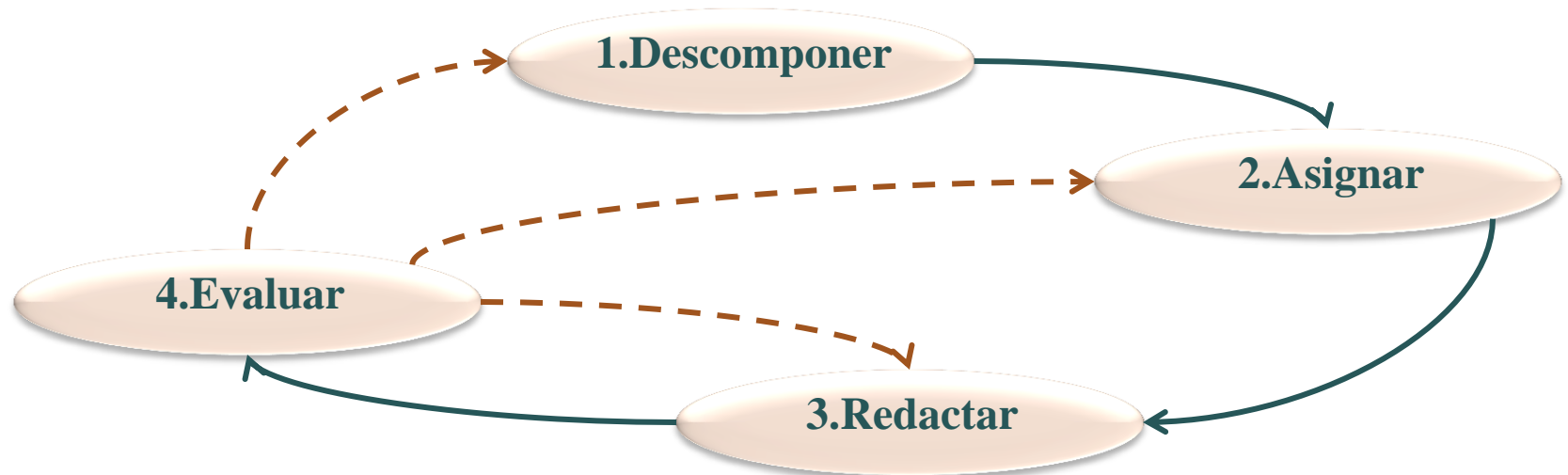
- Capítulo 7. Sección 7.4. J. Ortega, M. Anguita, A. Prieto. “Arquitectura de Computadores”. Thomson, 2005. ESII/C.1
ORT arq

➤ Complementaria

- Thomas Rauber, Gudula Rünger. “Parallel Programming: for Multicore and Cluster Systems.” Springer, 2010. Disponible en línea (biblioteca UGR): <http://dx.doi.org/10.1007/978-3-642-04818-0>
- Barry Wilkinson. “Parallel programming : techniques and applications using networked workstations and parallel computer”, 2005. ESIIT/D.1 WIL par

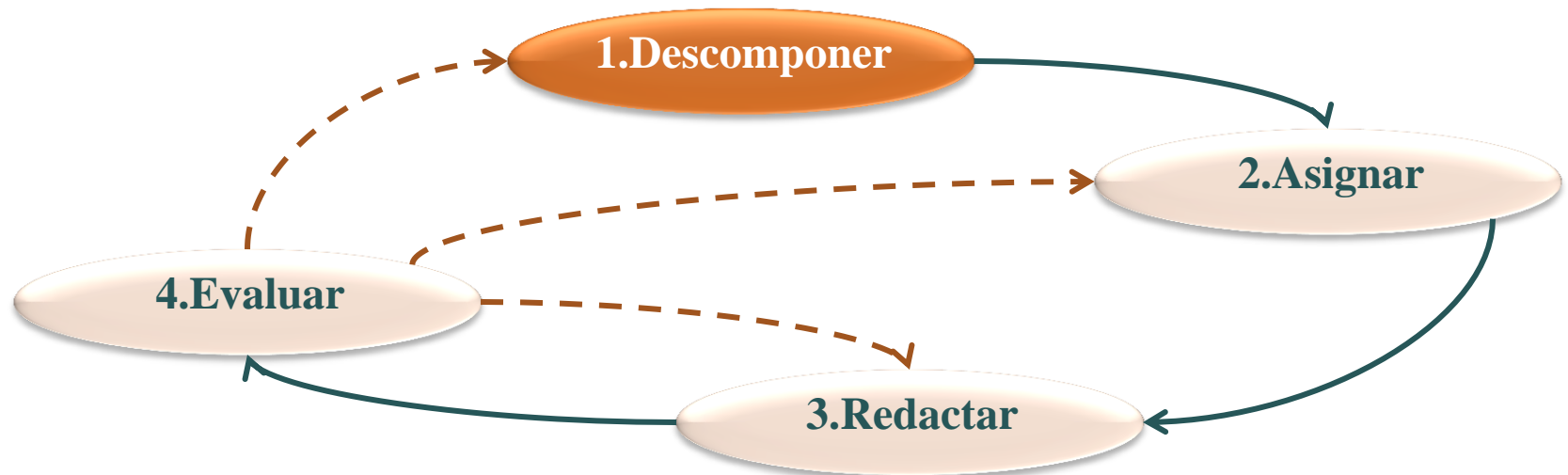
Proceso de paralelización

1. *Descomponer (descomposition)* en tareas independientes o Localizar paralelismo
2. *Asignar (planificar+mapear)* tareas a procesos y/o threads.
3. *Redactar* código paralelo.
4. *Evaluar* prestaciones.



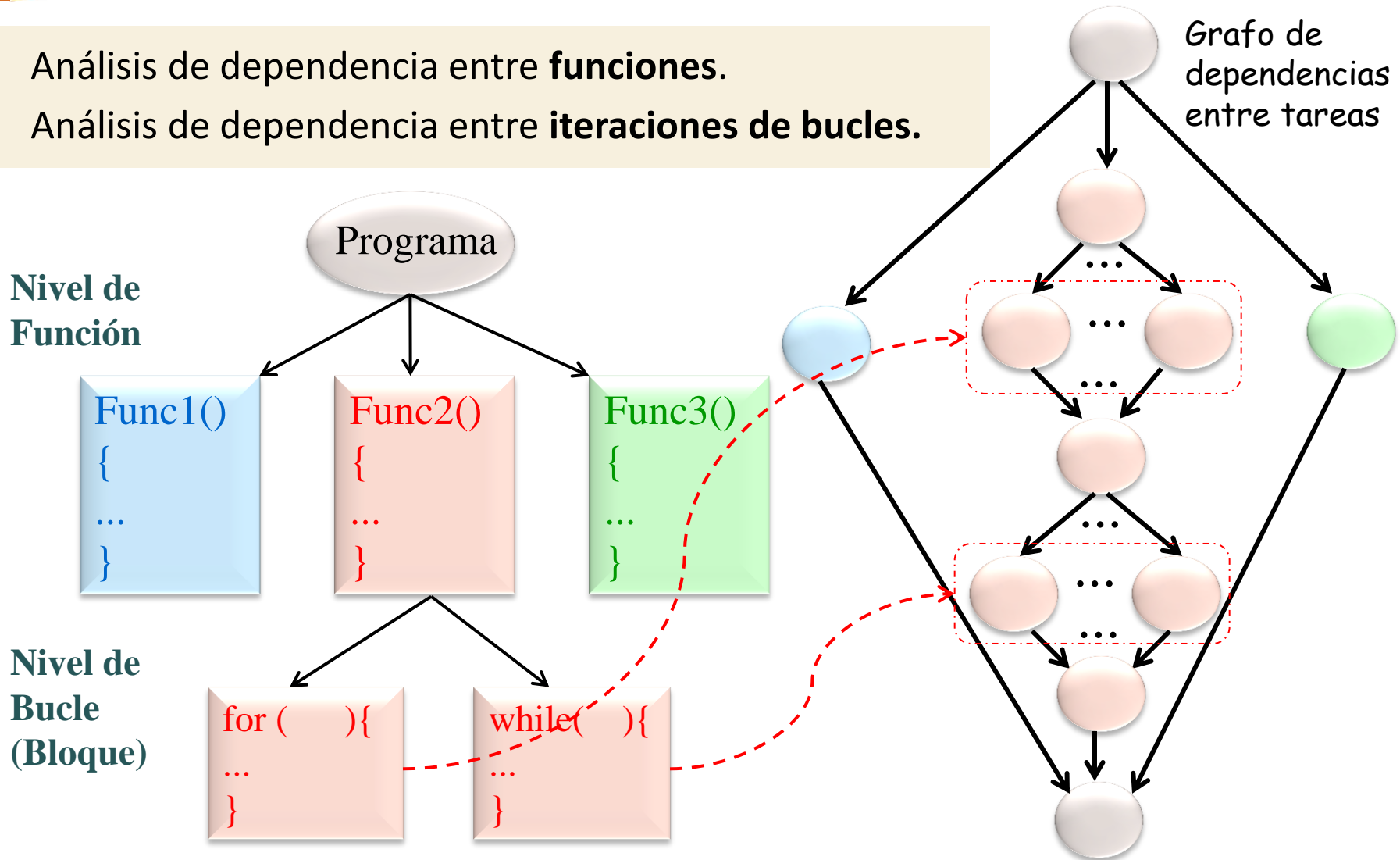
Proceso de paralelización

1. *Descomponer* en tareas independientes.
2. *Asignar* (*planificar+mapear*) tareas a procesos y/o threads.
3. *Redactar* código paralelo.
4. *Evaluar* prestaciones.



Descomposición en tareas independientes

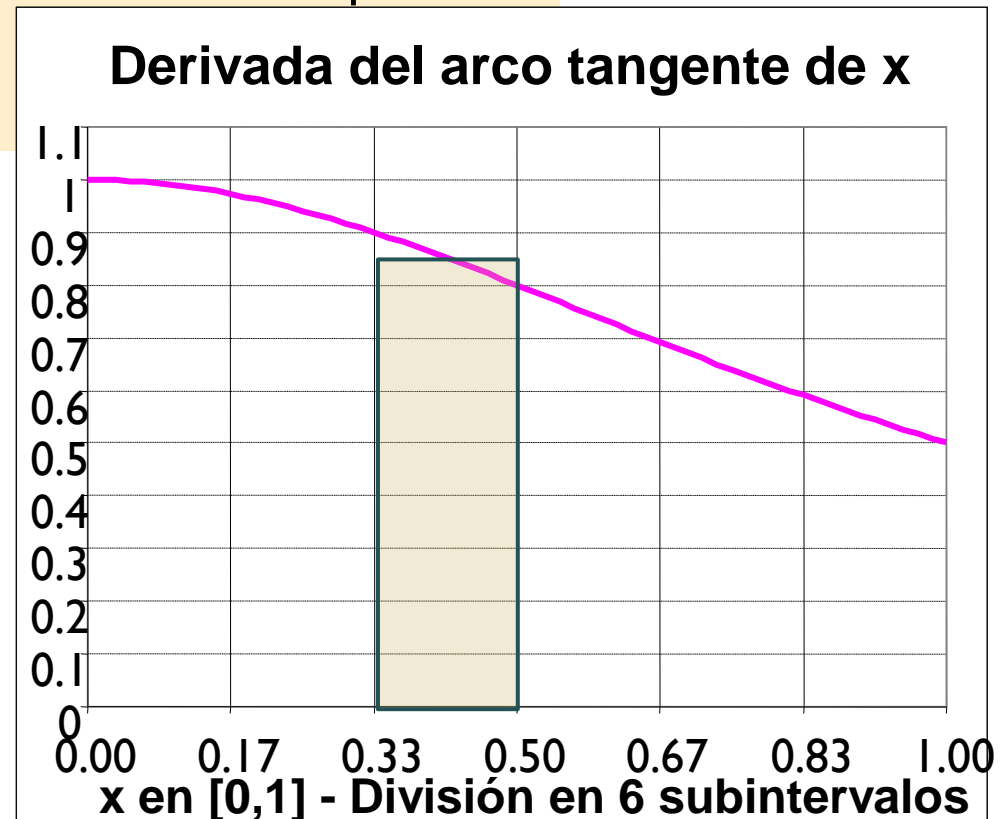
- Análisis de dependencia entre **funciones**.
- Análisis de dependencia entre **iteraciones de bucles**.



Ejemplo de cálculo PI: Descomposición en tareas independientes

$$\left. \begin{array}{l} \arctan'(x) = \frac{1}{1+x^2} \\ \arctan(1) = \frac{\pi}{4} \\ \arctan(0) = 0 \end{array} \right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \arctan(x) \Big|_0^1 = \frac{\pi}{4} - 0$$

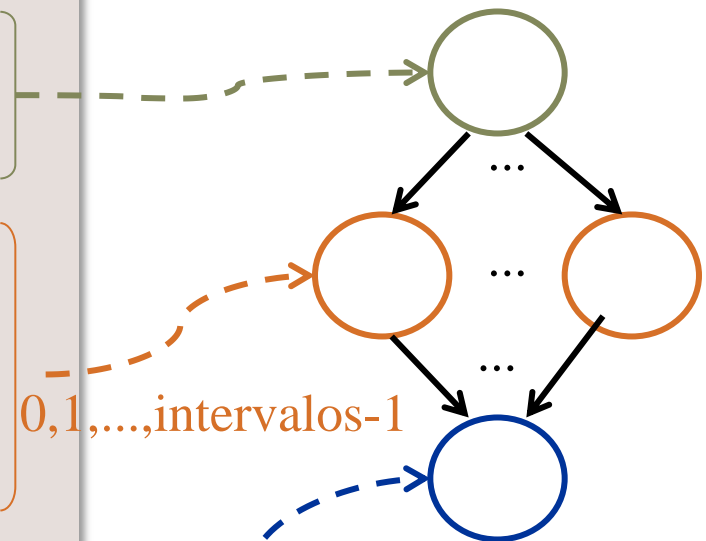
- PI se puede calcular por integración numérica.



Ejemplo de cálculo PI: Descomposición en tareas independientes

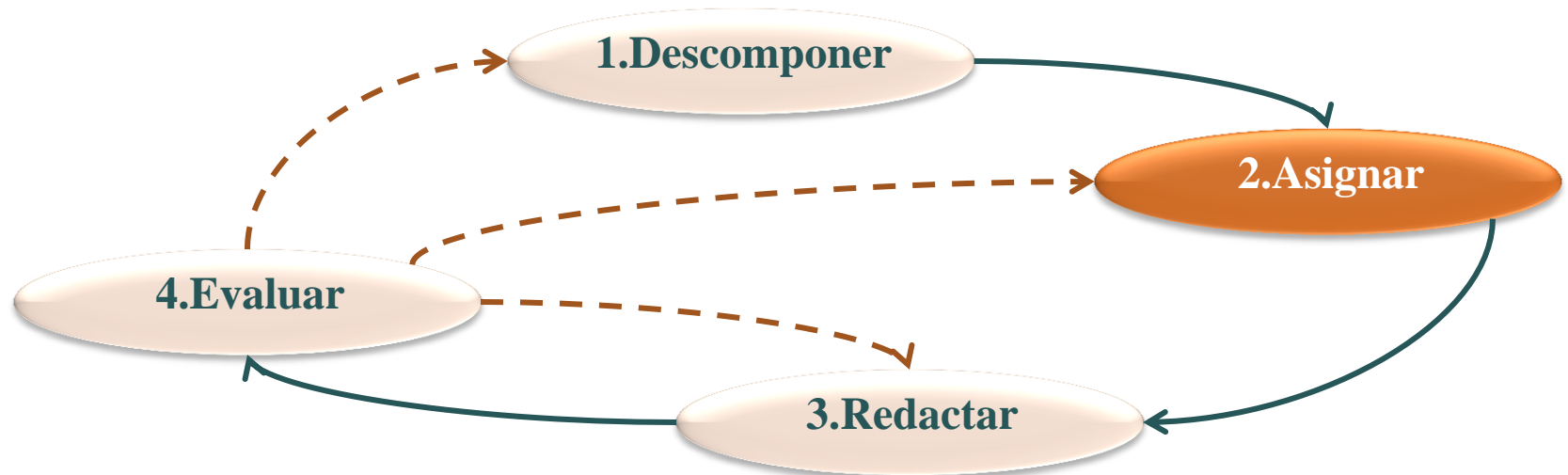
```
main(int argc, char **argv) {  
    double ancho, sum=0;  
    int intervalos, i;  
  
    intervalos = atoi(argv[1]);  
    ancho = 1.0/(double) intervalos;  
  
    for (i=0;i< intervalos; i++){  
        x = (i+0.5)*ancho;  
        sum = sum + 4.0/(1.0+x*x);  
    }  
  
    sum* = ancho;  
}
```

Grafo de dependencias entre tareas



Proceso de paralelización

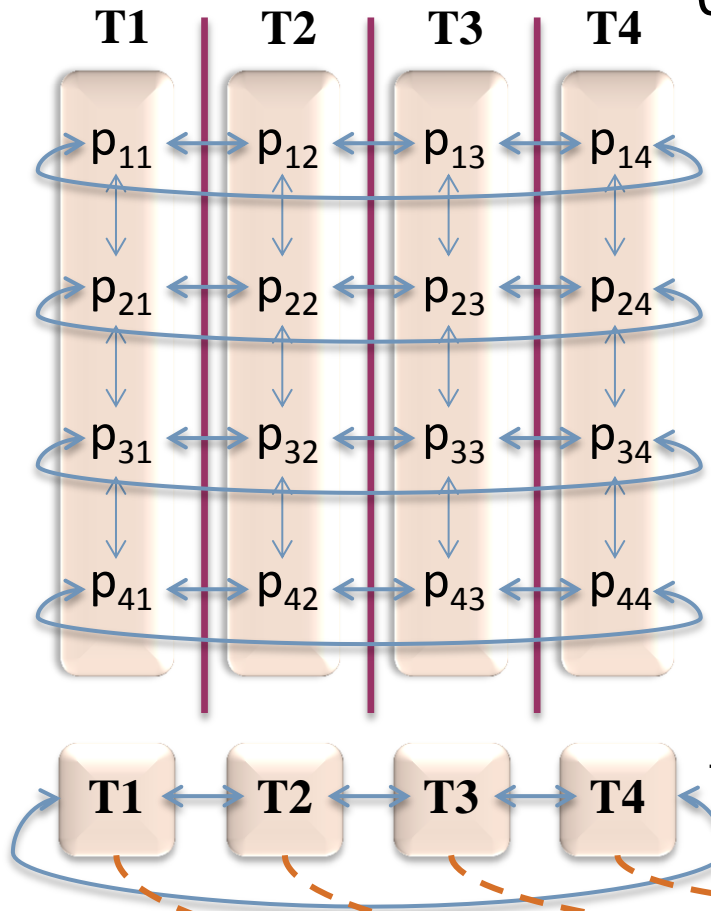
1. *Descomponer* en tareas independientes.
2. *Asignar (planificar+mapear)* tareas a procesos y/o threads.
3. *Redactar* código paralelo.
4. *Evaluar* prestaciones.



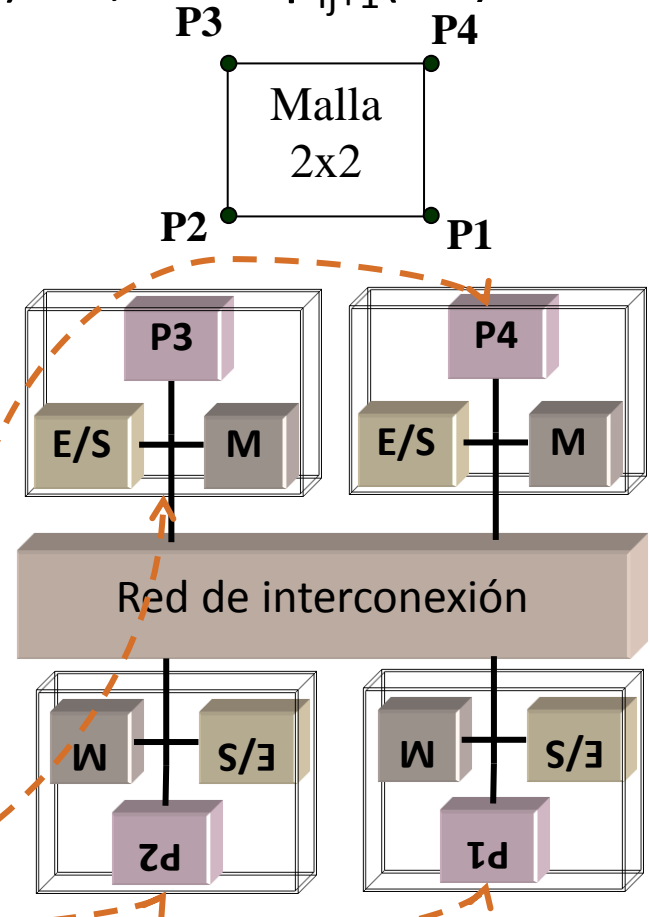
Asignación (planificación + mapeo). Ej.: filtrado de imagen I

$$p_{ij}(t) = 0,75 \times p_{ij}(t-1) + 0,0625 \times p_{i-1j}(t-1) + 0,0625 \times p_{ij-1}(t-1) + 0,0625 \times p_{i+1j}(t-1) + 0,0625 \times p_{ij+1}(t-1)$$

Planificación: agrupar tareas en threads





Mapeo: asignar threads a cores/procesadores



Asignación de tareas a procesos/threads I

- Incluimos: agrupación de tareas en procesos/threads (*scheduling*) y mapeo a procesadores/cores (*mapping*)
- La granularidad de la carga asignada a los procesos/threads depende de:
 - número de cores o procesadores o elementos de procesamiento
 - tiempo de comunicación/sincronización frente a tiempo de cálculo
- Equilibrado de la carga (tareas = código + datos) o *load balancing*:
 - Objetivo: unos procesos/threads no deben hacer esperar a otros

Asignación de tareas a procesos/threads II

- ¿De qué depende el equilibrado?
 - La arquitectura:
 - homogénea frente a heterogénea,
 - uniforme frente a no uniforme 
 - La aplicación/descomposición
- Tipos de asignación:
 - Estática
 - Está determinado qué tarea va a realizar cada procesador o core
 - Explícita: programador 
 - Implícita: herramienta de programación al generar el código ejecutable
 - Dinámica (en tiempo de ejecución)
 - Distintas ejecuciones pueden asignar distintas tareas a un procesador o core
 - Explícita: el programador
 - Implícita: herramienta de programación al generar el código ejecutable

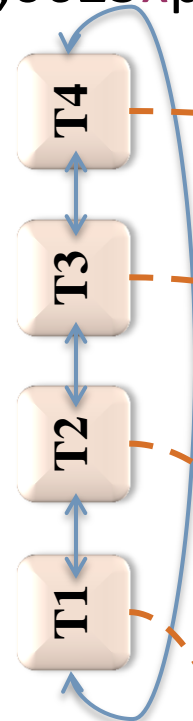
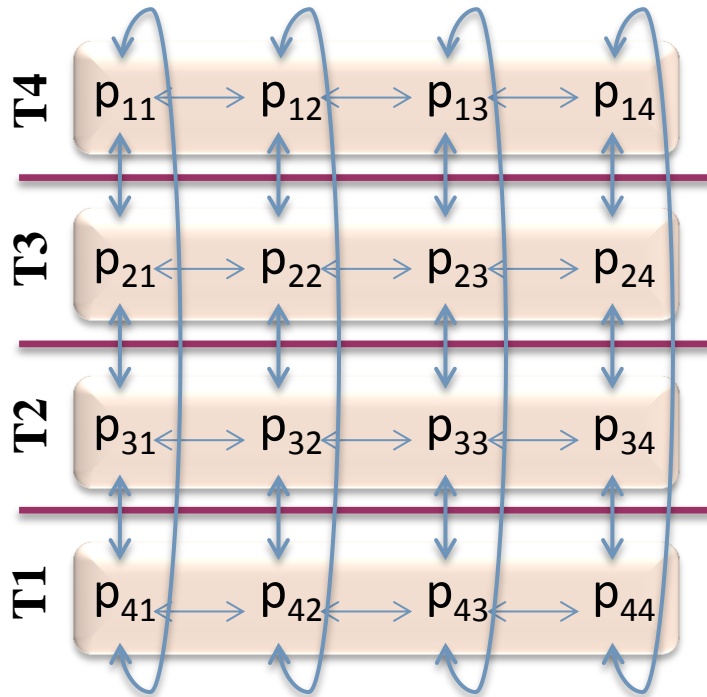
Mapeo de procesos/threads a unidades de procesamiento

- Normalmente se deja al SO el mapeo de threads (*light process*)
- Lo puede hacer el entorno o sistema en tiempo de ejecución (*runtime system*) de la herramienta de programación
- El programador puede influir

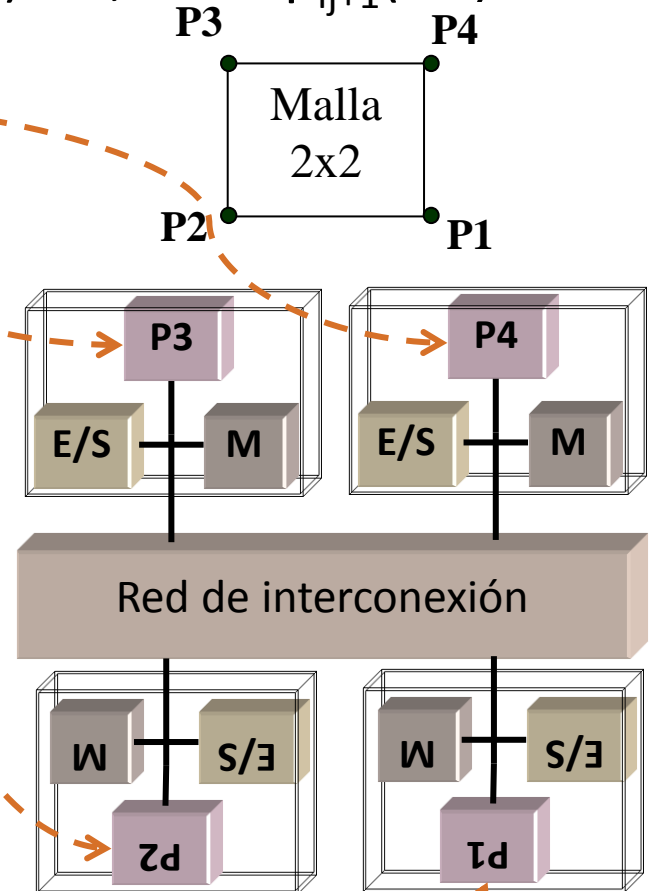
Asignación (planificación + mapeo). Ej.: filtrado de imagen II

$$p_{ij}(t) = 0,75 \times p_{ij}(t-1) + 0,0625 \times p_{i-1j}(t-1) + 0,0625 \times p_{ij-1}(t-1) + 0,0625 \times p_{i+1j}(t-1) + 0,0625 \times p_{ij+1}(t-1)$$

Planificación: agrupar tareas en threads



Mapeo: asignar threads a cores/procesadores



Códigos filtrado imagen

Descomposición por columnas

```
#include <omp.h>
...
omp_set_num_threads(M)
#pragma omp parallel private(i)
{
    for (i=0;i<N;i++) {
        #pragma omp for
        for (j=0;j<M;j++) {
            pS[i,j] = 0,75*p[i,j] +
                0,0625*(p[i-1,j]+p[i,j-1]+
                    p[i+1,j]+ p[i,j+1]);
        }
    }
}
...
```

Descomposición por filas

```
#include <omp.h>
...
omp_set_num_threads(N)
#pragma omp parallel private(j)
{
    #pragma omp for
    for (i=0;i<N;i++) {
        for (j=0;j<M;j++) {
            pS[i,j] = 0,75*p[i,j] +
                0,0625*(p[i-1,j]+p[i,j-1]+
                    p[i+1,j]+ p[i,j+1]);
        }
    }
}
...
```


Asignación. Ej.: multiplicación matriz por vector I

$$c = A \bullet b; \quad c_i = \sum_{k=0}^{M-1} a_{ik} \bullet b_k = a_i^T \bullet b, \quad c(i) = \sum_{k=0}^{M-1} A(i,k) \bullet b(k), \quad i = 0, \dots, N-1$$

$ \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} \\ a_{61} & a_{62} & a_{63} & a_{64} & a_{65} & a_{66} \\ a_{71} & a_{72} & a_{73} & a_{74} & a_{75} & a_{76} \\ a_{81} & a_{82} & a_{83} & a_{84} & a_{85} & a_{86} \end{pmatrix} $	$ \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} $	$ \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \\ c_8 \end{pmatrix} $
$ \bullet $		$ = $
		$ \begin{aligned} c(1) &= \sum_{k=0}^{M-1} A(1,k) \bullet b(k) \\ c(2) &= \sum_{k=0}^{M-1} A(2,k) \bullet b(k) \\ c(3) &= \sum_{k=0}^{M-1} A(3,k) \bullet b(k) \\ c(4) &= \sum_{k=0}^{M-1} A(4,k) \bullet b(k) \\ c(5) &= \sum_{k=0}^{M-1} A(5,k) \bullet b(k) \\ c(6) &= \sum_{k=0}^{M-1} A(6,k) \bullet b(k) \\ c(7) &= \sum_{k=0}^{M-1} A(7,k) \bullet b(k) \\ c(8) &= \sum_{k=0}^{M-1} A(8,k) \bullet b(k) \end{aligned} $

8x6

Asignación. Ej.: multiplicación matriz por vector II

$$c = A \bullet b; \quad c_i = \sum_{k=0}^{M-1} a_{ik} \bullet b_k = a_i^T \bullet b, \quad c(i) = \sum_{k=0}^{M-1} A(i,k) \bullet b(k), \quad i = 0, \dots, N-1$$

Diagram illustrating matrix-vector multiplication:

Matrix A (8x6) multiplied by vector b (6x1) to produce vector c (8x1).

Matrix A elements: $a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, \dots, a_{81}, a_{82}, a_{83}, a_{84}, a_{85}, a_{86}$.

Vector b elements: $b_1, b_2, b_3, b_4, b_5, b_6$.

Resulting vector c elements: $c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8$.

Dimensions: $N=8$, $M=6$.

Equation: $A \bullet b = c$

Row 1 calculation: $c_1 = a_{11}b_1 + a_{12}b_2 + a_{13}b_3 + a_{14}b_4 + a_{15}b_5 + a_{16}b_6$

Row 2 calculation: $c_2 = a_{21}b_1 + a_{22}b_2 + a_{23}b_3 + a_{24}b_4 + a_{25}b_5 + a_{26}b_6$

Row 3 calculation: $c_3 = a_{31}b_1 + a_{32}b_2 + a_{33}b_3 + a_{34}b_4 + a_{35}b_5 + a_{36}b_6$

Row 4 calculation: $c_4 = a_{41}b_1 + a_{42}b_2 + a_{43}b_3 + a_{44}b_4 + a_{45}b_5 + a_{46}b_6$

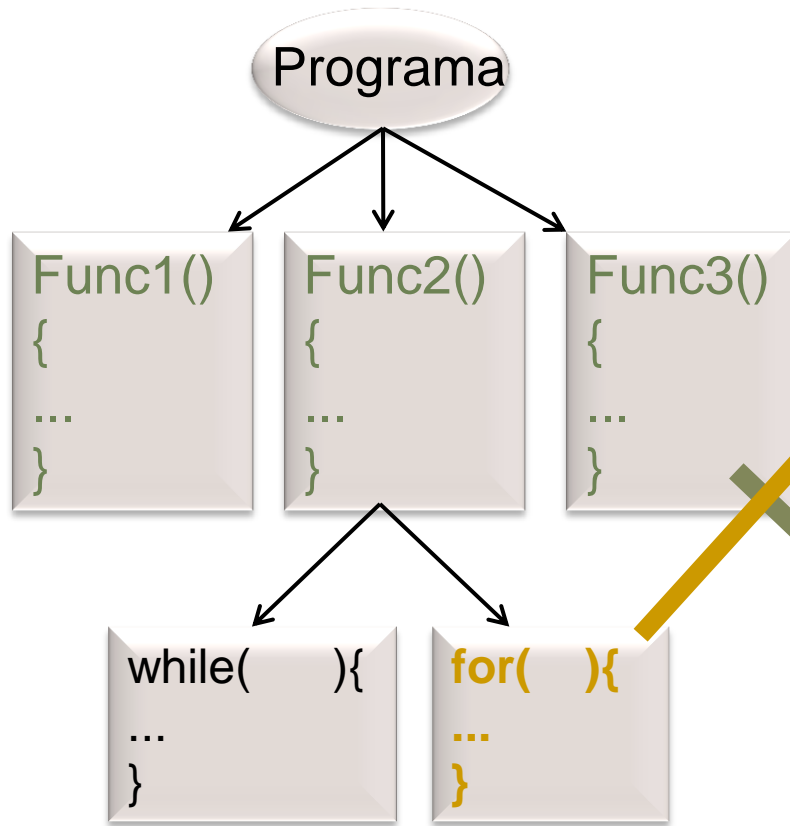
Row 5 calculation: $c_5 = a_{51}b_1 + a_{52}b_2 + a_{53}b_3 + a_{54}b_4 + a_{55}b_5 + a_{56}b_6$

Row 6 calculation: $c_6 = a_{61}b_1 + a_{62}b_2 + a_{63}b_3 + a_{64}b_4 + a_{65}b_5 + a_{66}b_6$

Row 7 calculation: $c_7 = a_{71}b_1 + a_{72}b_2 + a_{73}b_3 + a_{74}b_4 + a_{75}b_5 + a_{76}b_6$

Row 8 calculation: $c_8 = a_{81}b_1 + a_{82}b_2 + a_{83}b_3 + a_{84}b_4 + a_{85}b_5 + a_{86}b_6$

Ejemplo de asignación estática del paralelismo de tareas y datos con OpenMP



```
Func1() {...}
Func2() { ...
    #pragma omp parallel for \
        schedule(static)
    for (i=0;i<N;i=++) {
        código para i
    }
...}
Func3() {...}
Main () {
    #pragma omp parallel sections
    { #pragma omp section
        Func1();
      #pragma omp section
        Func2();
      #pragma omp section
        Func3();
    }
...}
```

Asignación estática

- Asignación **estática** y explícita de las iteraciones de un bucle:

Bucle

```
for (i=0;i<Iter;i++) {  
    código para i  
}
```

Estática Round-Robin

```
for (i=idT;i<Iter;i=i+nT) {  
    código para i  
}
```

Estática Continua

```
for (i= idT* $\frac{Iter}{nT}$  ; i< (idT+1)* $\frac{Iter}{nT}$  ; i++) {  
    código para i  
}
```

Asignación dinámica

- Asignación **dinámica** y explícita de las iteraciones de un bucle:

Bucle

```
for (i=0;i<Iter;i++) {  
    código para i    }
```



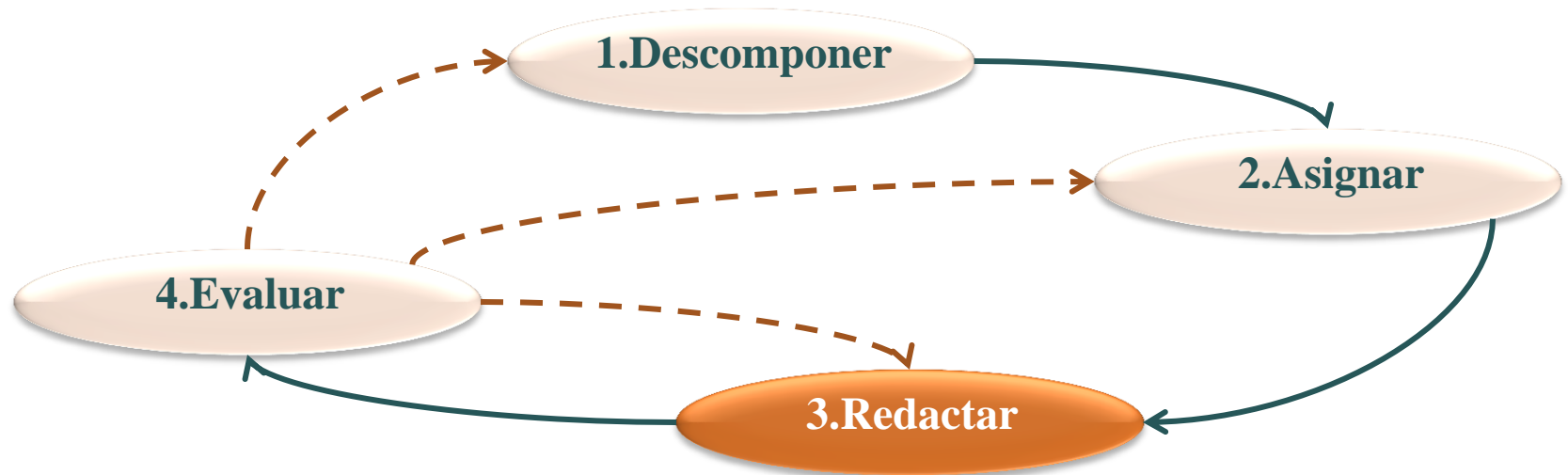
Dinámica

```
lock(k) ;  
    n=i;   i=i+1;  
unlock(k) ;  
while (n<Iter) {  
    código para n ;  
  
    lock(k) ;  
        n=i;   i=i+1;  
    unlock(k) ;  
  
}
```

NOTA: La variable *i* se supone inicializada a 0

Proceso de paralelización

1. *Descomponer* en tareas independientes.
2. *Asignar* (*planificar+mapear*) tareas a procesos y/o threads.
3. *Redactar* código paralelo.
4. *Evaluar* prestaciones.



Ejemplo: cálculo de PI con OpenMP/C

```
#include <omp.h>
#define NUM_THREADS 4
main(int argc, char **argv) {
    long double ancho, x, sum=0;  int intervalos, i;
    intervalos = atoi(argv[1]);
    ancho = 1.0/(double) intervalos;
    omp_set_num_threads(NUM_THREADS);
    #pragma omp parallel
    {
        #pragma omp for reduction(+:sum) private(x) \
            schedule(dynamic)
        for (i=0; i< intervalos; i++) {
            x = (i+0.5)*ancho;  sum = sum + 4.0/(1.0+x*x);
        }
    }
    sum* = ancho;
}
```

Crear/Terminar

Comunicar/sincronizar

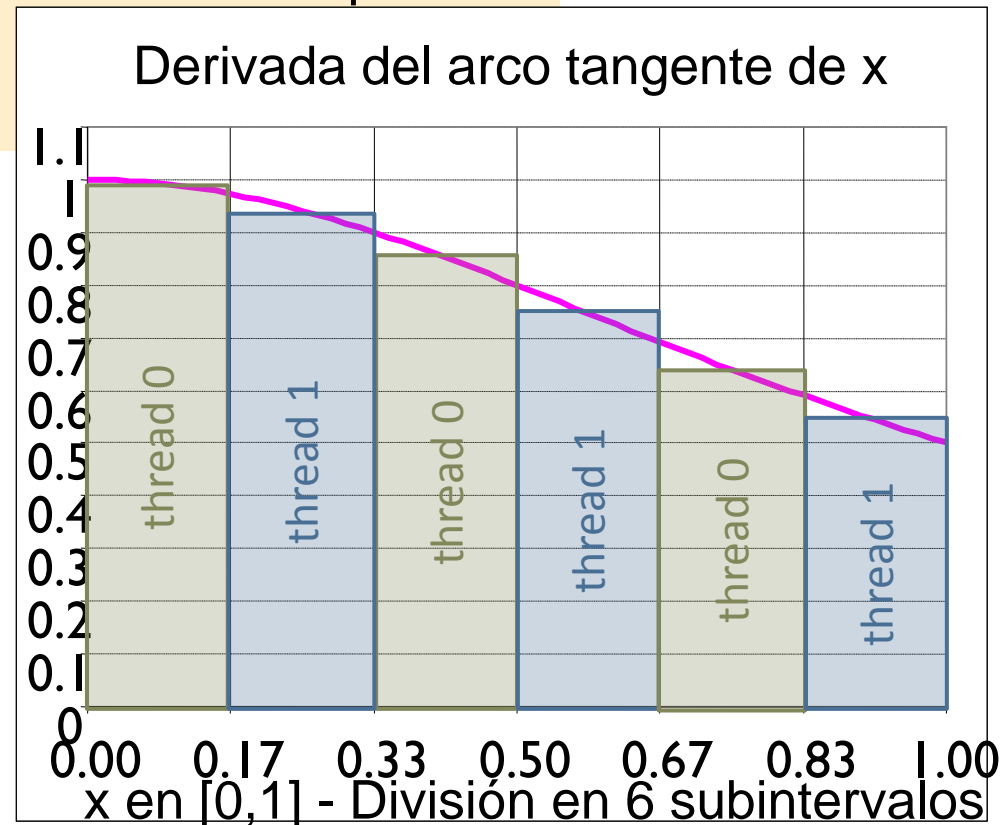
Agrupar/Asignar

Localizar

Asignación de tareas a 2 threads estática por turno rotatorio

$$\left. \begin{array}{l} \arctan'(x) = \frac{1}{1+x^2} \\ \arctan(1) = \frac{\pi}{4} \\ \arctan(0) = 0 \end{array} \right\} \Rightarrow \int_0^1 \frac{1}{1+x^2} = \arctan(x) \Big|_0^1 = \frac{\pi}{4} - 0$$

- PI se puede calcular por integración numérica.



Ejemplo: cálculo de PI en MPI/C

```
#include <mpi.h>
main(int argc, char **argv) {
    double ancho,x,lsum,sum; int intervalos,i,nproc,iproc;
    if (MPI_Init(&argc, &argv) != MPI_SUCCESS) exit(1);
    MPI_Comm_size(MPI_COMM_WORLD, &nproc);
    MPI_Comm_rank(MPI_COMM_WORLD, &iproc);
    intervalos=atoi(argv[1]);
    ancho=1.0/(double) intervalos; lsum=0;
    for (i=iproc; i<intervalos; i+=nproc) {
        x = (i+0.5)*ancho; lsum+= 4.0/(1.0+x*x);
    }
    lsum*= ancho;
    MPI_Reduce(&lsum, &sum, 1, MPI_DOUBLE,
               MPI_SUM,0,MPI_COMM_WORLD);
    MPI_Finalize();
}
```

Enrolar

Localizar-Agrupar

Comunicar/sincronizar

Desenrolar

Proceso de paralelización

1. *Descomponer* en tareas independientes.
2. *Asignar (agrupar+mapear)* tareas a procesos y/o threads.
3. *Redactar* código paralelo.
4. *Evaluar* prestaciones.

