ARQUITECTURA DE COMPUTADORES II

AUTORES:
David Expósito Singh
Florin Isaila
Daniel Higuero Alonso-Mardones
Javier García Blas
Borja Bergua Guerra

Área de Arquitectura y Tecnología de Computadores
Departamento de Informática
Universidad Carlos III de Madrid

Julio de 2012

TEMA 2:
PROGRAMACIÓN PARALELA (II)

Índice

- Ejemplos de Aplicaciones Paralelizables
- Etapas del Proceso de Paralelización:
  - Descomposición
  - Asignación
  - Orquestación
  - Mapeo
- Ejemplo: Simulación de Corrientes Oceánicas

Índice

- Ejemplos de Aplicaciones Paralelizables

- Etapas del Proceso de Paralelización
  - Descomposición
  - Asignación
  - Orquestación
  - Mapeo

- Ejemplo: Simulación de Corrientes Oceánicas

Programas paralelos

- Necesidades del arquitecto de computadores:

  Para entender y evaluar las decisiones de diseño en máquinas paralelas es necesario entender el tipo de aplicaciones que requiere de estas máquinas.

- Necesidades del programador:
  - La programación paralela es una actividad compleja.
  - Es importante alcanzar buen rendimiento.
  - Metodología para realizar programas paralelos.

Definiciones

- ***Tarea:***
  - *Unidad de trabajo* en el cómputo paralelo.
  - Se ejecuta secuencialmente, la concurrencia es solo entre tareas.
  - Tareas de *grano fino* vs. tareas de *grano grueso.*
- ***Proceso****:*
  - Entidad abstracta que realiza las tareas asignadas a los procesadores.
  - Los procesos se comunican y sincronizan para realizar las tareas.
- ***Procesador:***
  - Máquina física sobre la que se ejecuta un proceso.

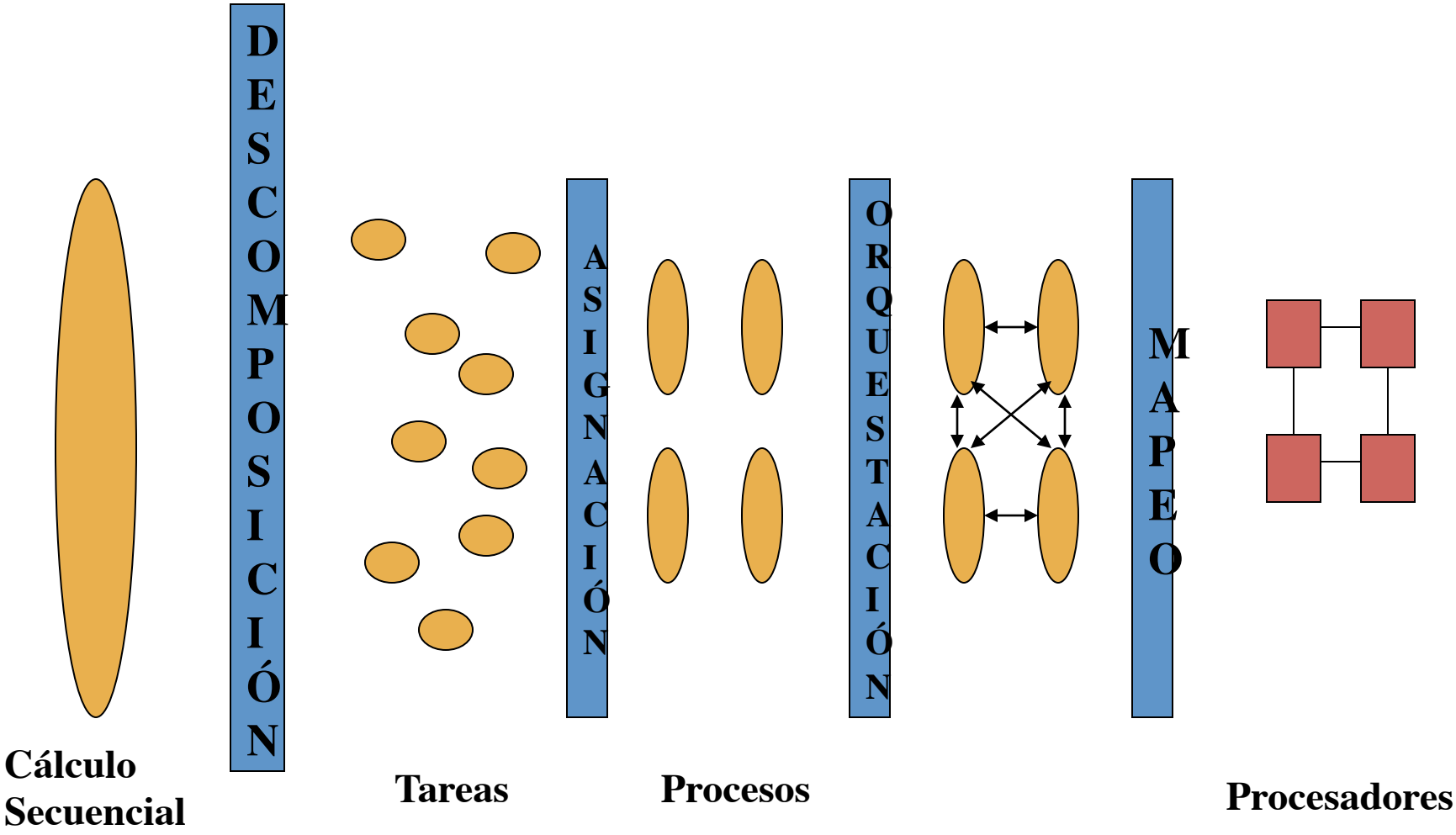
Índice

- Ejemplos de Aplicaciones Paralelizables

- Etapas del Proceso de Paralelización
  - Descomposición
  - Asignación
  - Orquestación
  - Mapeo

- Ejemplo: Simulación de Corrientes Oceánicas

4 etapas en el proceso de paralelización [Culler99]



Etapa 1
Descomposición

- Descomposición del problema en pequeñas *tareas*.

- Metas:
  - Cuanto más pequeñas (grano fino) mejor.
  - Deben representar cantidades similares de trabajo.
  - Sin redundancia de cómputo ni almacenamiento.
  - El número de éstas debe "escalar" con el tamaño del problema.

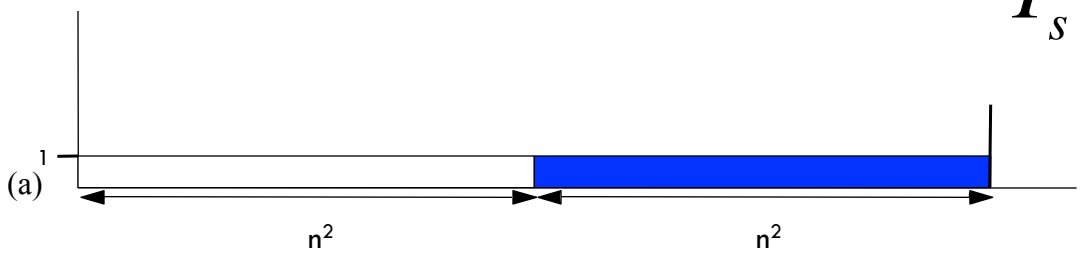
- Límite de la concurrencia: *Ley de Amdahl*.

¿Hasta dónde es posible paralelizar?

Ejemplo: Programa con dos Fases

1. Operación independiente que debe ser llevada a cabo en una matriz de dimensión n x n
2. Se suma los n^2 valores calculados en la fase anterior

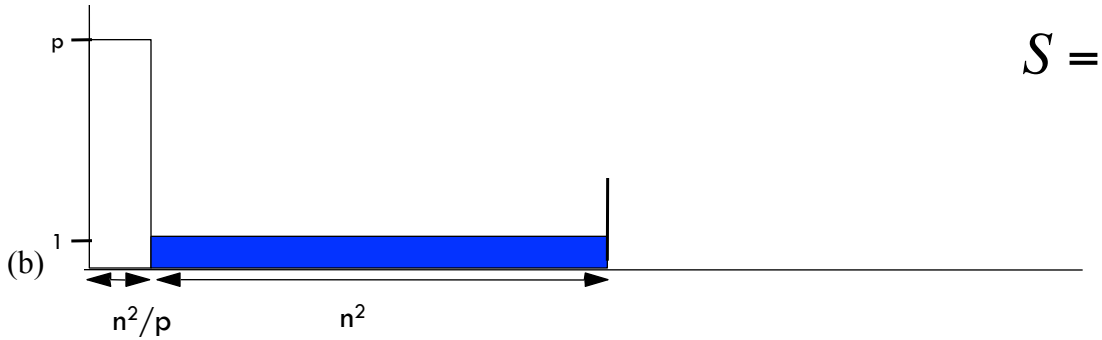
$$T_s = 2n^2$$



Primer intento de paralelización

1. Se paraleliza la primera fase: p: grado de concurrencia
2. La segunda es secuencial

$$T_p = \frac{n^2}{p} + n^2$$



$$S = \frac{2n^2}{\frac{n^2}{p} + n^2} = \frac{2p}{p+1}$$

Segundo intento de paralelización

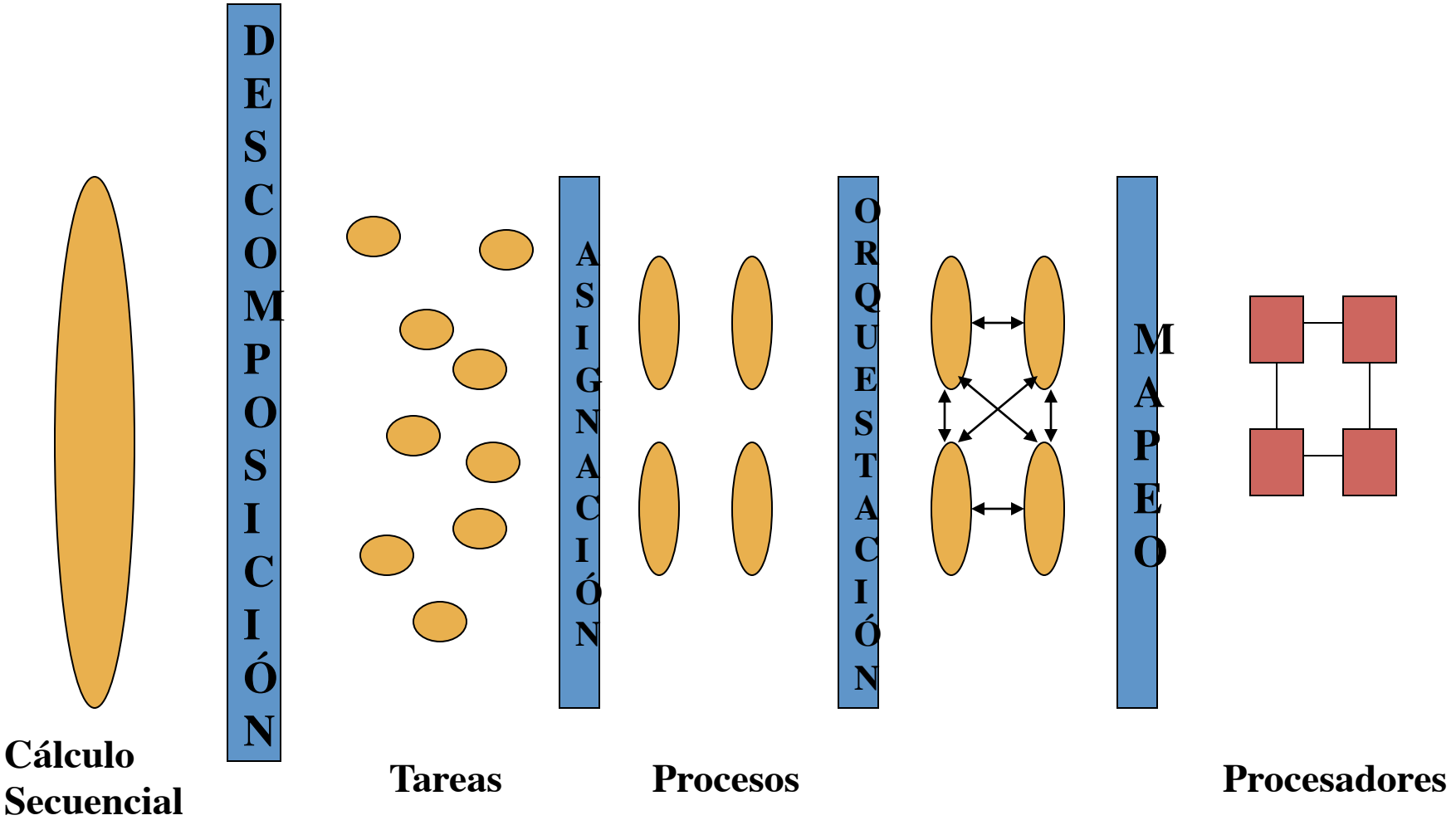
En la segunda fase: sumemos en cada procesador los elementos de su submatriz y luego hagamos la suma total

$$T_p = \frac{n^2}{p} + \frac{n^2}{p} + p$$

$$S = \frac{p \times 2n^2}{(2n^2 + p^2)}$$

Ahora la aceleración es casi lineal!!

4 etapas en el proceso de paralelización [Culler99]



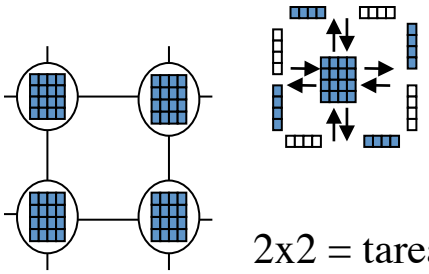
Etapa 2
Asignación

- Especificar el mecanismo por el cual las tareas serán distribuidas entre los procesos.

  Ejemplos:

  ¿Qué proceso tiene la responsabilidad de calcular las fuerzas de ciertas estrellas?

  ¿Qué zona de un grid conviene asignar a un proceso?

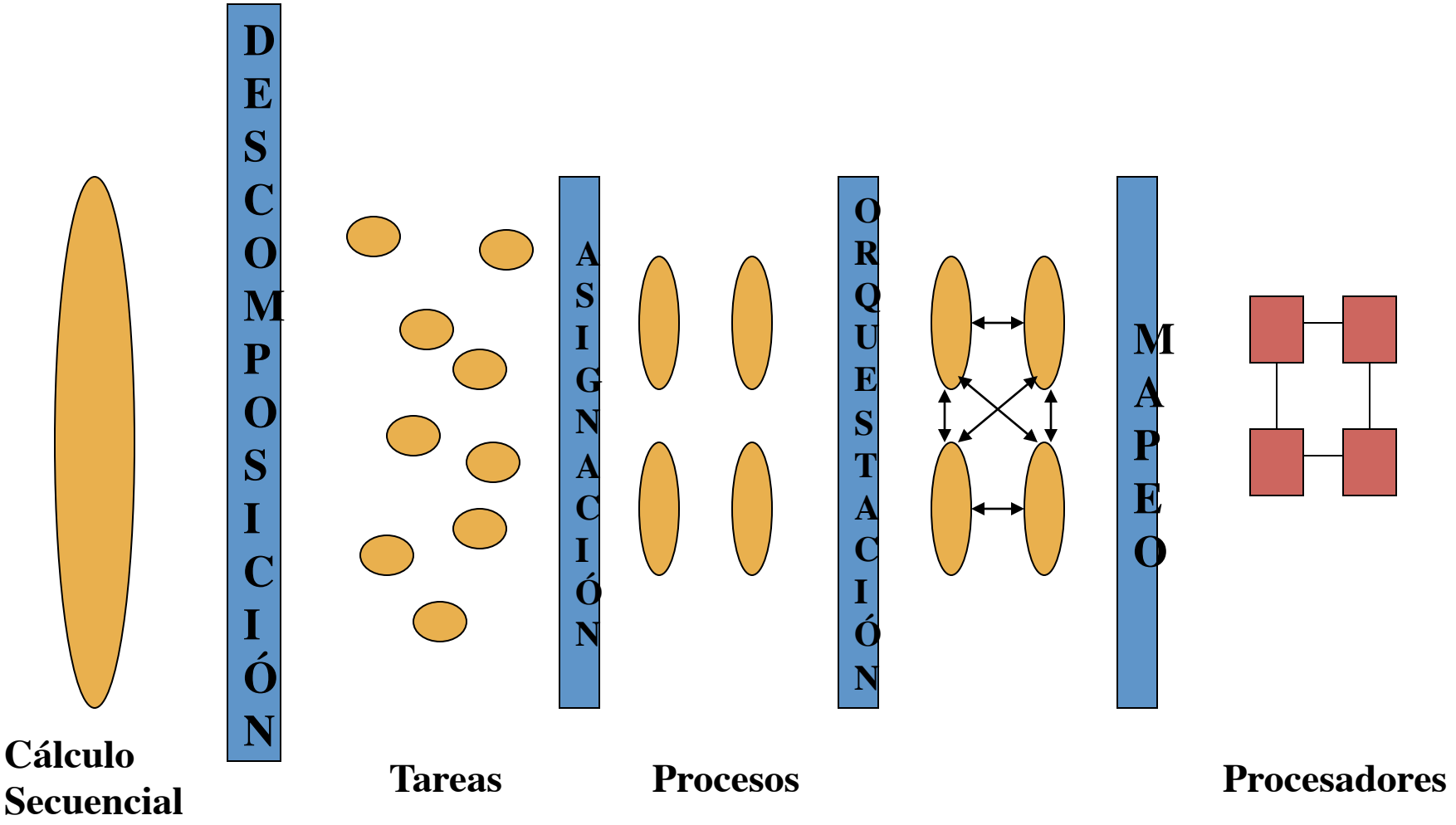


2x2 = tareas
4x4 = 16 comunicaciones
16 x 4 = 64 valores transmitidos

Etapa 2
Asignación

- Aspectos a cuidar:
  - Distribución a partir de datos requiere de técnicas de *equilibrado de carga.*
  - Distribución a partir de funciones (actividades) requiere de *algoritmos de planificación.*
  - Reducir las comunicaciones.

4 etapas en el proceso de paralelización [Culler99]



Etapa 3
Orquestación

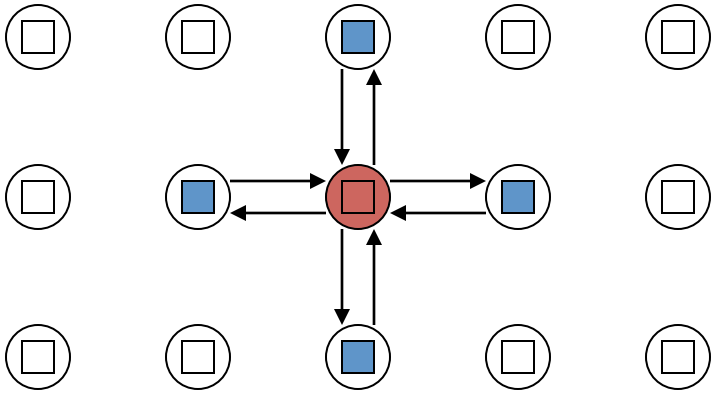
- ¿Qué debe hacerse?
  - Establecer mecanismos para nombrar y manejar datos.
  - Estructurar la comunicación entre los procesos.
  - Sincronización.

Las características de esta etapa dependen del modelo de programación con el que se trabaje.

Patrones de comunicación

1. local/global.
2. estructurado/no-estructurado.
3. estático/dinámico.
4. síncrono/asíncrono.

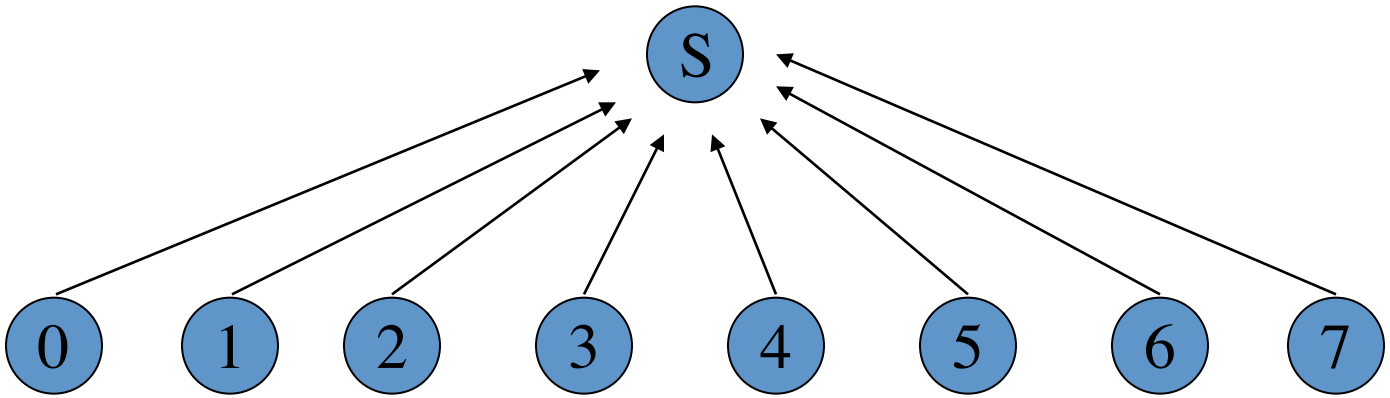
Comunicación Local



$$X_{i,j}^{t+1} = \frac{4X_{i,j}^{t} + X_{i-1,j}^{t} + X_{i+1,j}^{t} + X_{i,j-1}^{t} + X_{i,j+1}^{t}}{8}$$

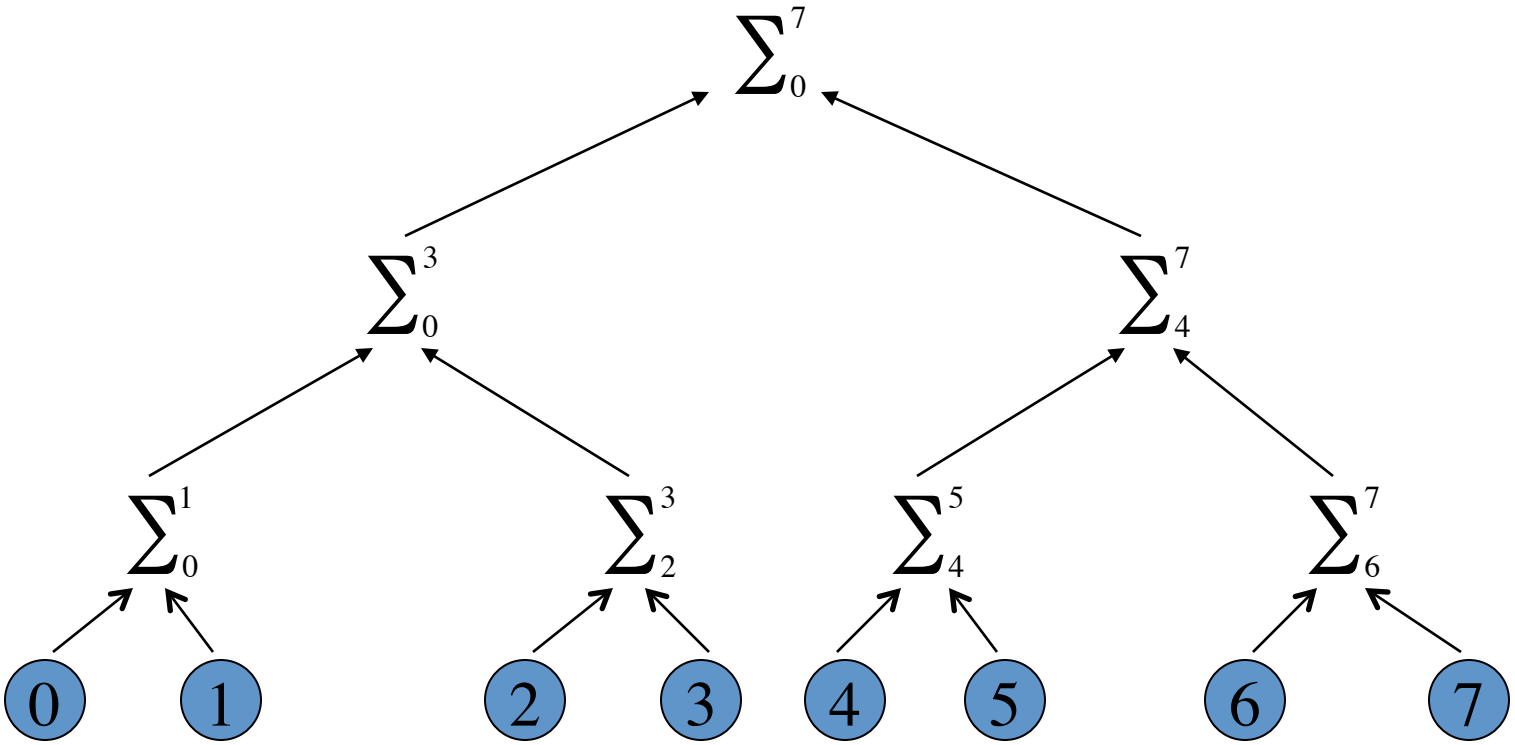
for t=0 to T-1
  send $X_{i,j}$ a cada vecino
  receive $X_{i-1,j}$ $X_{i+1,j}$ $X_{i,j-1}$ $X_{i,j+1}$ de los vecinos
  calcular $X_{i,j}$ para t+1
endfor

Comunicación Global



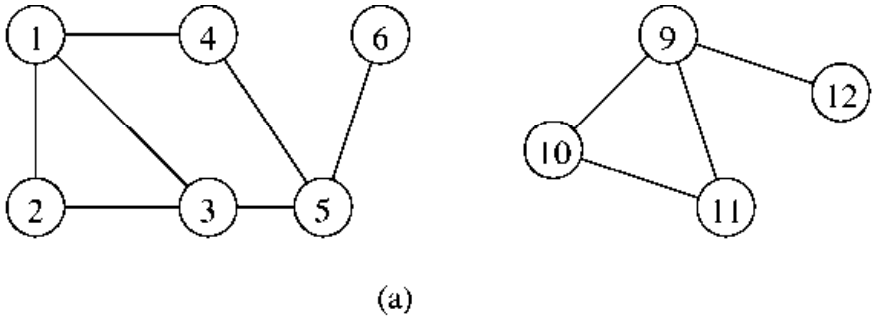
$$S = \sum_{i=0}^{N-1} X_i$$

Comunicación Estructurada

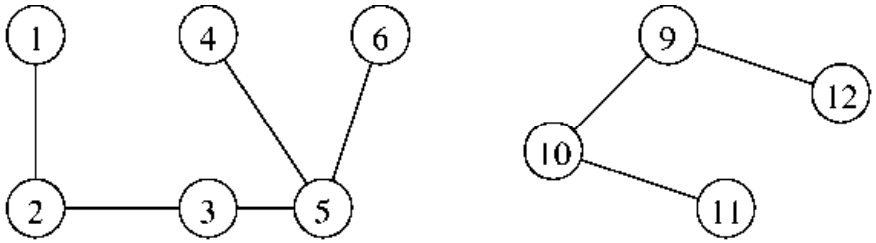


Comunicación no estructurada

Cálculo de componentes conexas

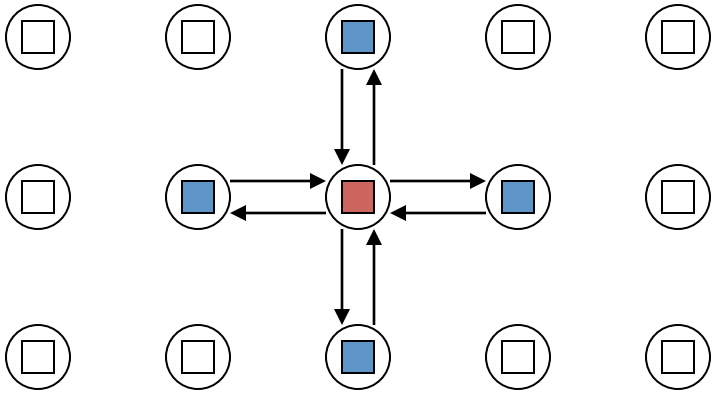


(a)



(b)

Comunicación Estática

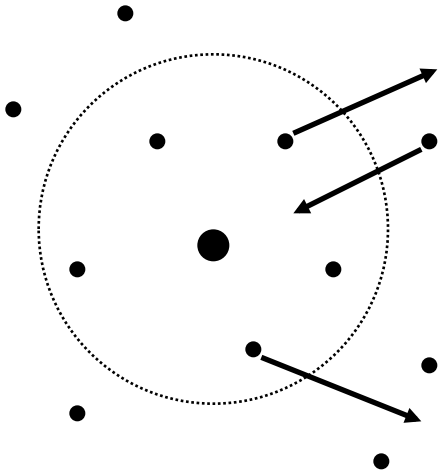


$$X_{i,j}^{t+1} = \frac{4X_{i,j}^{t} + X_{i-1,j}^{t} + X_{i+1,j}^{t} + X_{i,j-1}^{t} + X_{i,j+1}^{t}}{8}$$

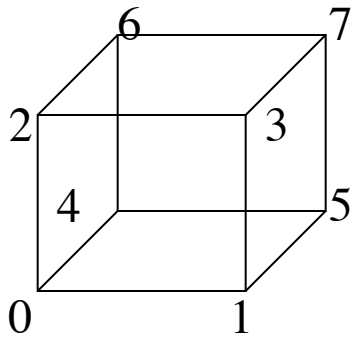
for t=0 to T-1
  send $X_{i,j}$ a cada vecino
  receive $X_{i-1,j}$ $X_{i+1,j}$ $X_{i,j-1}$ $X_{i,j+1}$ de los vecinos
  calcular $X_{i,j}$ para t+1
endfor

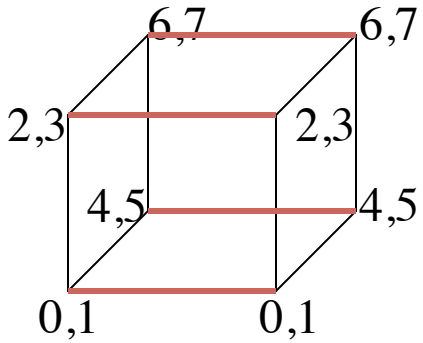
Comunicación Dinámica

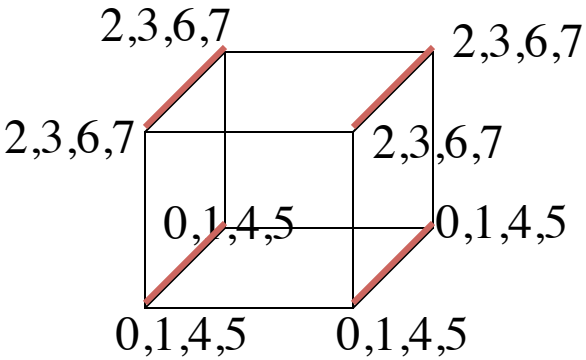
- N-body problems:
  - Evolución de las Galaxias.
  - Dinámica Molecular.

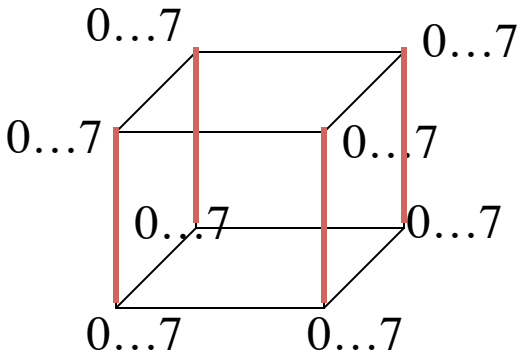


Comunicación Síncrona (*Broadcast*)

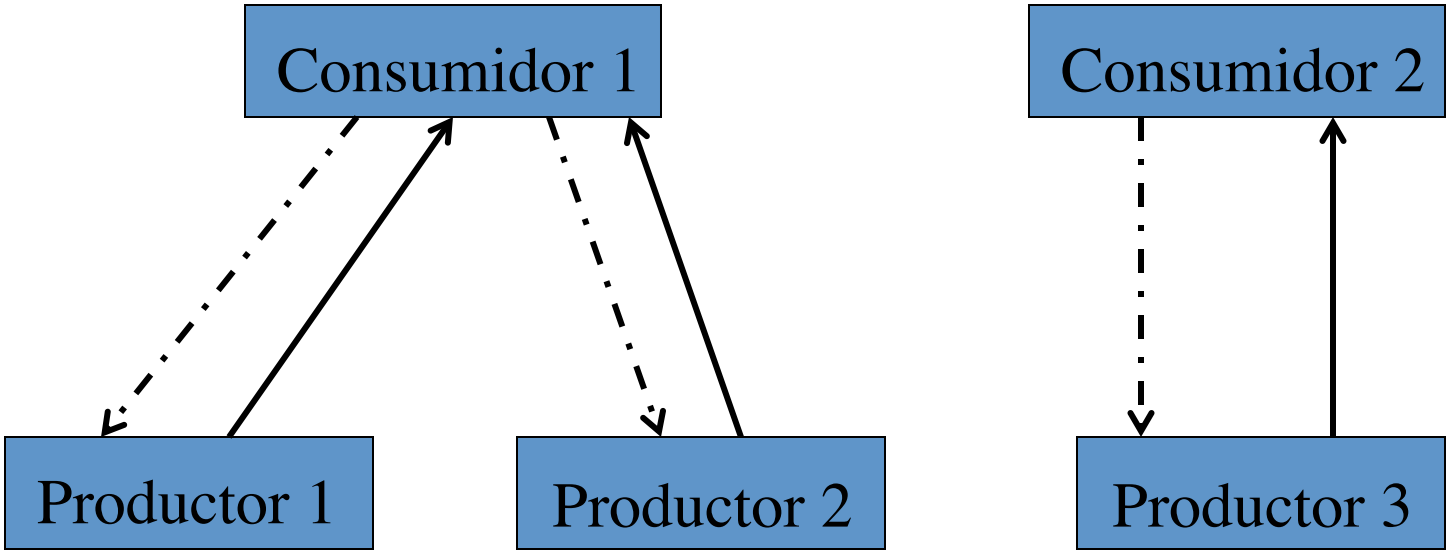




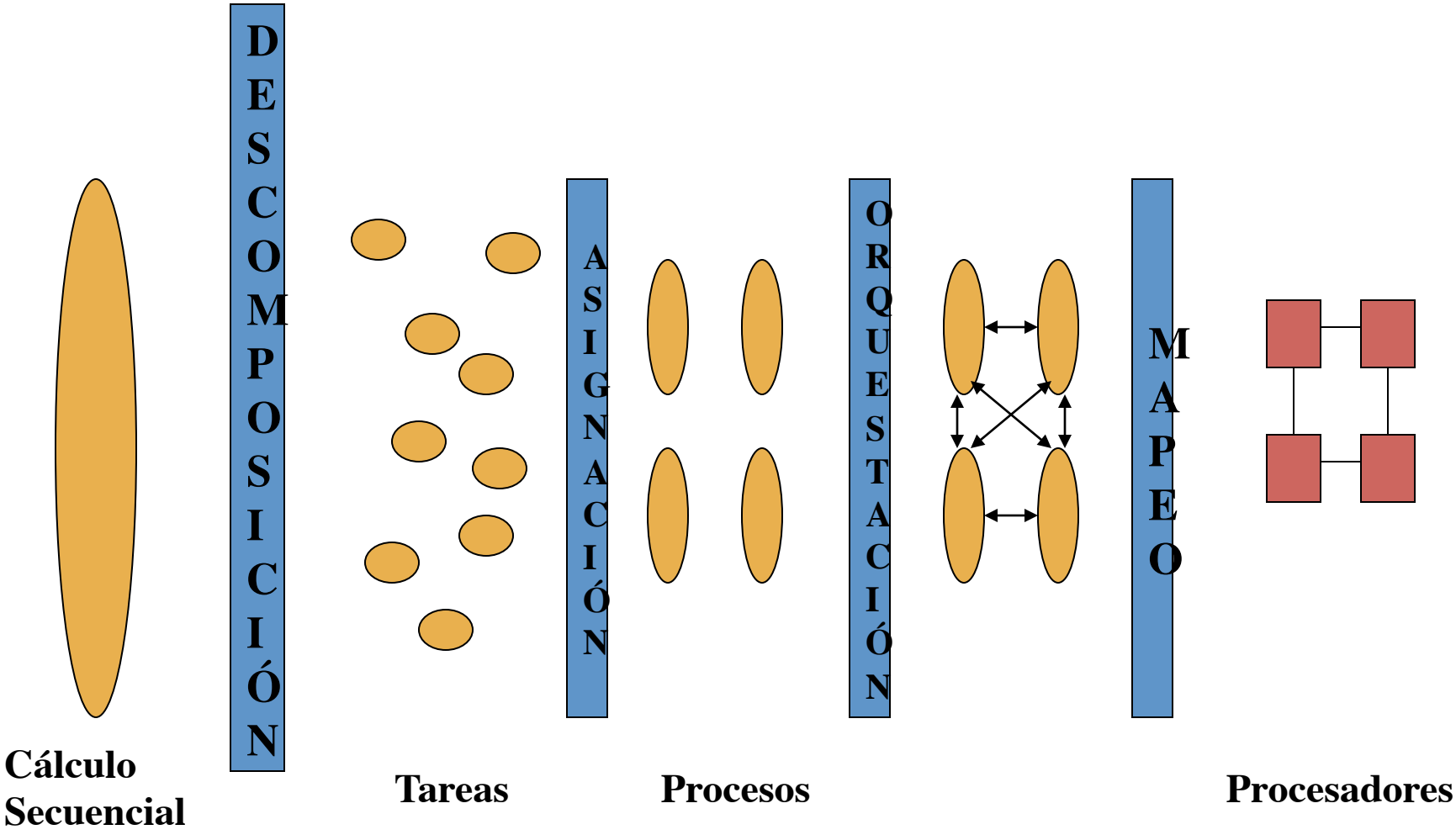




Comunicación Asíncrona



4 etapas en el proceso de paralelización [Culler99]



Etapa 4
Mapeo

- Asignación de proceso/hilo a procesadores.

- Especificar dónde se ejecutará cada proceso/hilo.

- Se espera minimizar el tiempo total de ejecución.
  1. Asignando tareas que ejecutan concurrentemente en diferentes procesadores.

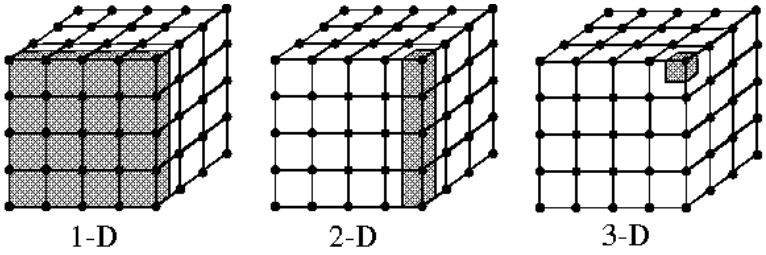
  2. Asignando tareas que comunican frecuentemente en un mismo procesador.

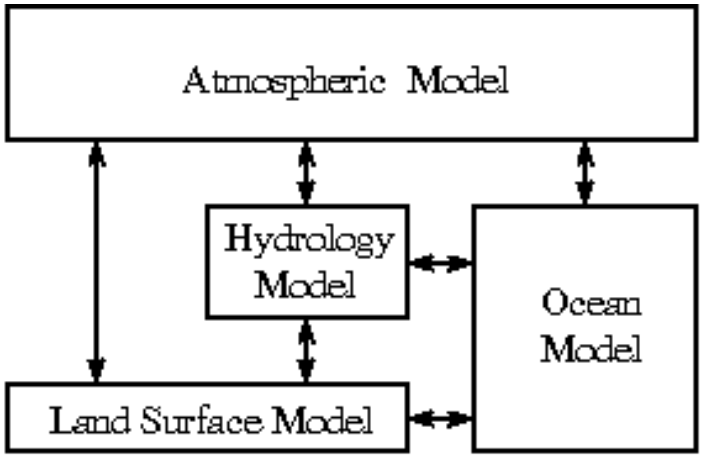
Consideraciones especiales a tener en cuenta en todas las etapas

- ¿Hay posibilidad de crear y destruir tareas de manera dinámica?

- Cuando se gestione información centralizada se debe cuidar la posibilidad de tener un *cuello de botella*.

Paralelizando Datos vs. Cálculo

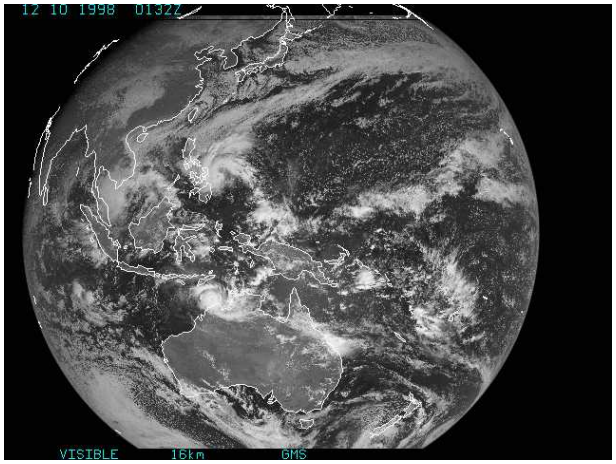


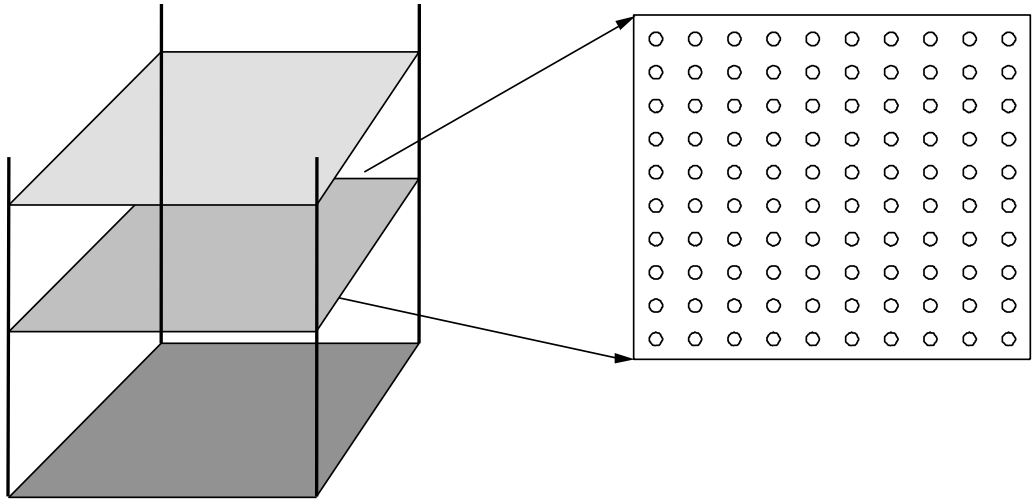


EJEMPLO

[Culler99]

Simulación de Corrientes Oceánicas



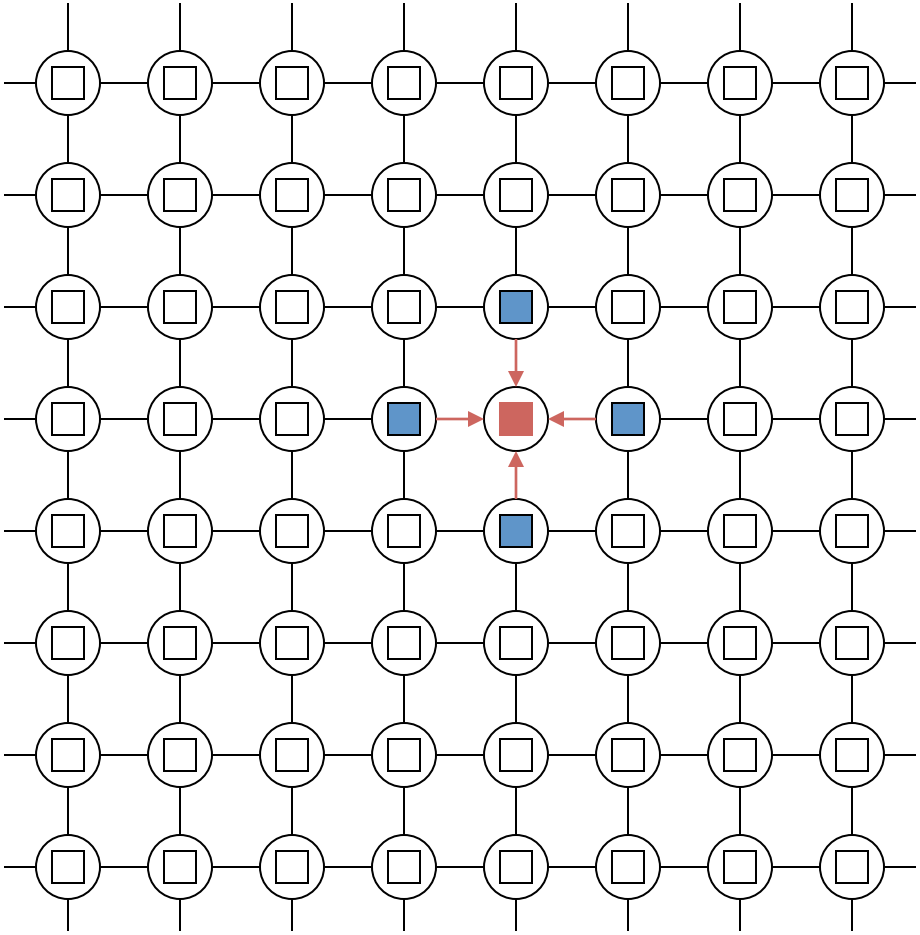


(a) Cross sections

(b) Spatial discretization of a cross section

- Modelado como *grid* (malla) de dos dimensiones.
  - Discretización en espacio y tiempo.
  - Grano fino temporal y espacial => mayor precisión.
- Muchas operaciones concurrentes.
- Modelo estático y regular.

Ecuación iterativa



Versión simplificada de la simulación de corrientes oceánicas

$$A[i,j] = 0{,}2 \times (A[i,j] + A[i, j-1] + A[i, j+1] + A[i-1, j] + A[i+1, j])$$

Ecuación iterativa

Gauss-Seidel:

- N x N puntos internos.
- Actualizaciones *in situ.*
- Al finalizar cada iteración se acumula las diferencias parciales.
- Se itera hasta alcanzar convergencia.

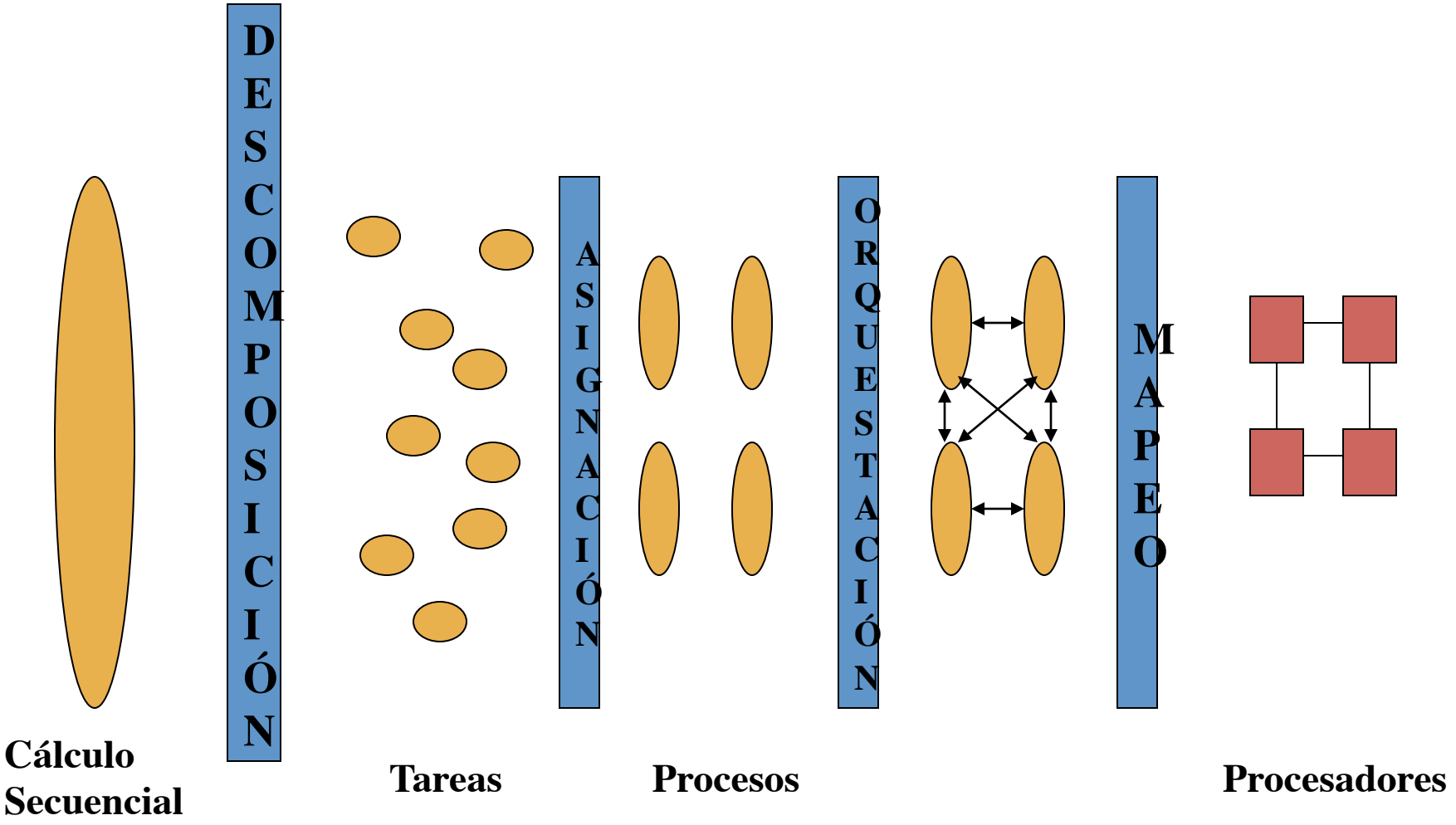
Versión Secuencial

```
1.  int n;                                  /*size of matrix: (n + 2-by-n + 2) elements*/
2.  float **A, diff = 0;

3.  main()
4.  begin
5.     read(n) ;                            /*read input parameter: matrix size*/
6.     A ← malloc (a 2-d array of size n + 2 by n + 2 doubles);
7.     initialize(A);                       /*initialize the matrix A somehow*/
8.     Solve (A);                           /*call the routine to solve equation*/
9.  end main

10. procedure Solve (A)                     /*solve the equation system*/
11.    float **A;                           /*A is an (n + 2)-by-(n + 2) array*/
12. begin
13.    int i, j, done = 0;
14.    float diff = 0, temp;
15.    while (!done) do                     /*outermost loop over sweeps*/
16.       diff = 0;                         /*initialize maximum difference to 0*/
17.       for i ← 1 to n do                 /*sweep over nonborder points of grid*/
18.          for j ← 1 to n do
19.             temp = A[i,j];             /*save old value of element*/
20.             A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                A[i,j+1] + A[i+1,j]); /*compute average*/
22.             diff += abs(A[i,j] - temp);
23.          end for
24.       end for
25.       if (diff/(n*n) < TOL) then done = 1;
26.    end while
27. end procedure
```

Las 4 fases

- Las fases de Descomposición y Asignación serán básicamente las mismas para los tres modelos de programación.
- La fase de Orquestación será diferente dependiendo del modelo de programación.

4 etapas en el proceso de paralelización [Culler99]



Descomposición

- Método Gauss-Seidel:

  - Estructura de bucles.

    Las iteraciones no son independientes entre sí. Por lo tanto, no puede utilizarse: **dependencia de los nuevos valores hacia la *izquierda* y hacia *arriba*.**

  - Análisis de dependencia de datos.

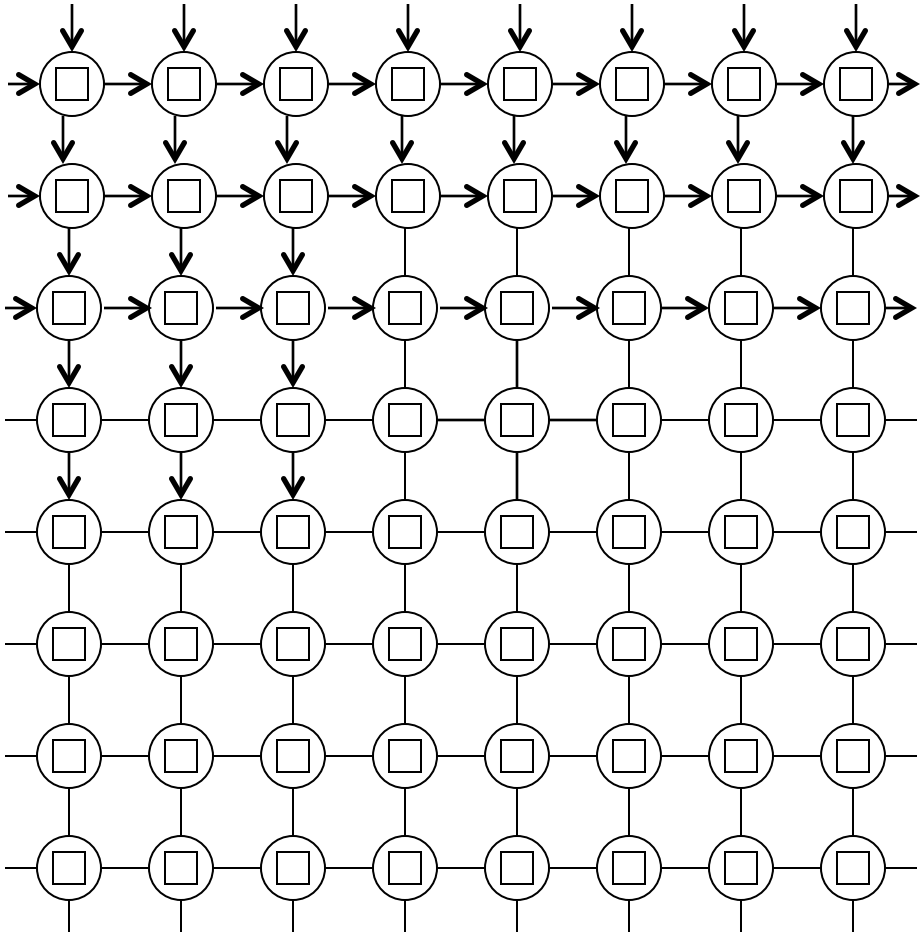
    Reestructurar las iteraciones sobre anti-diagonales

    Las anti-diagonales se pueden ejecutar concurrentemente

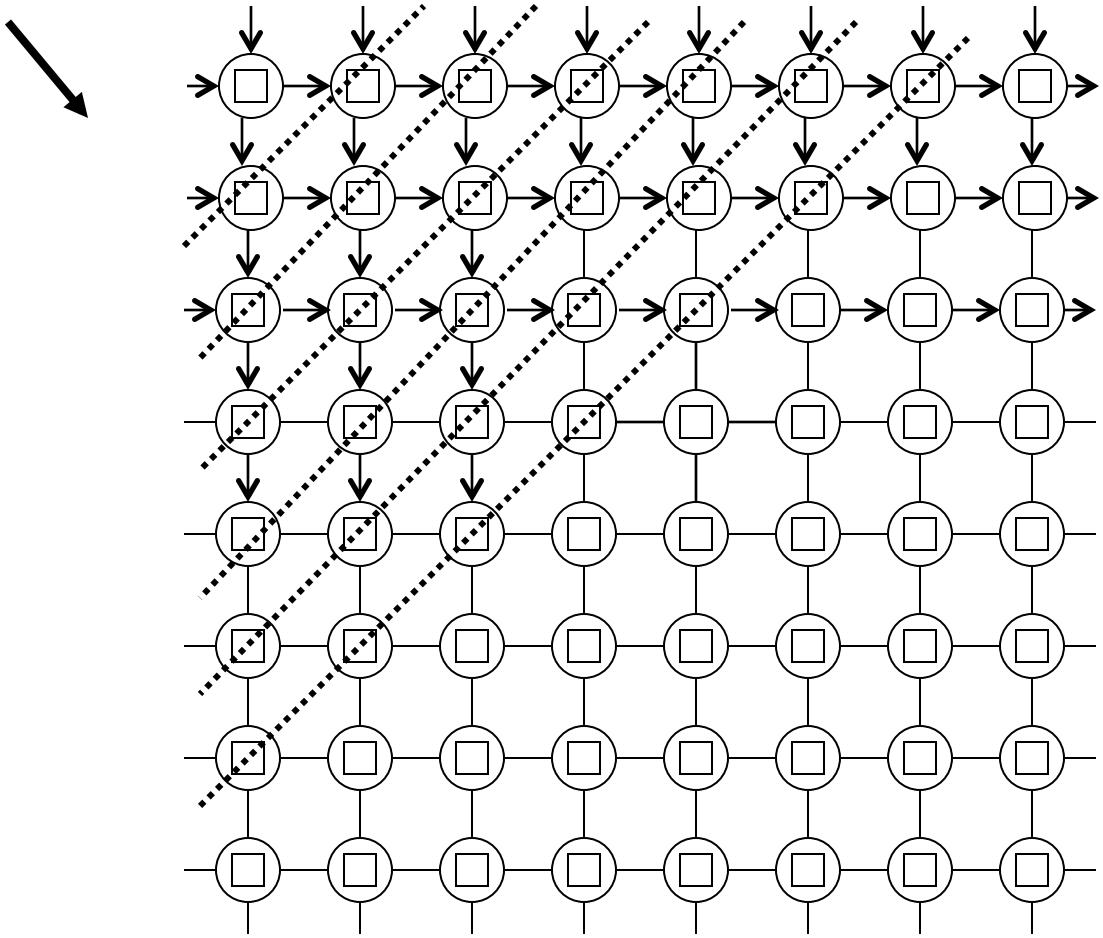
    Sincronización global entre iteraciones

    Causa desequilibrio

Dependencia y concurrencia para el método Gauss-Seidel



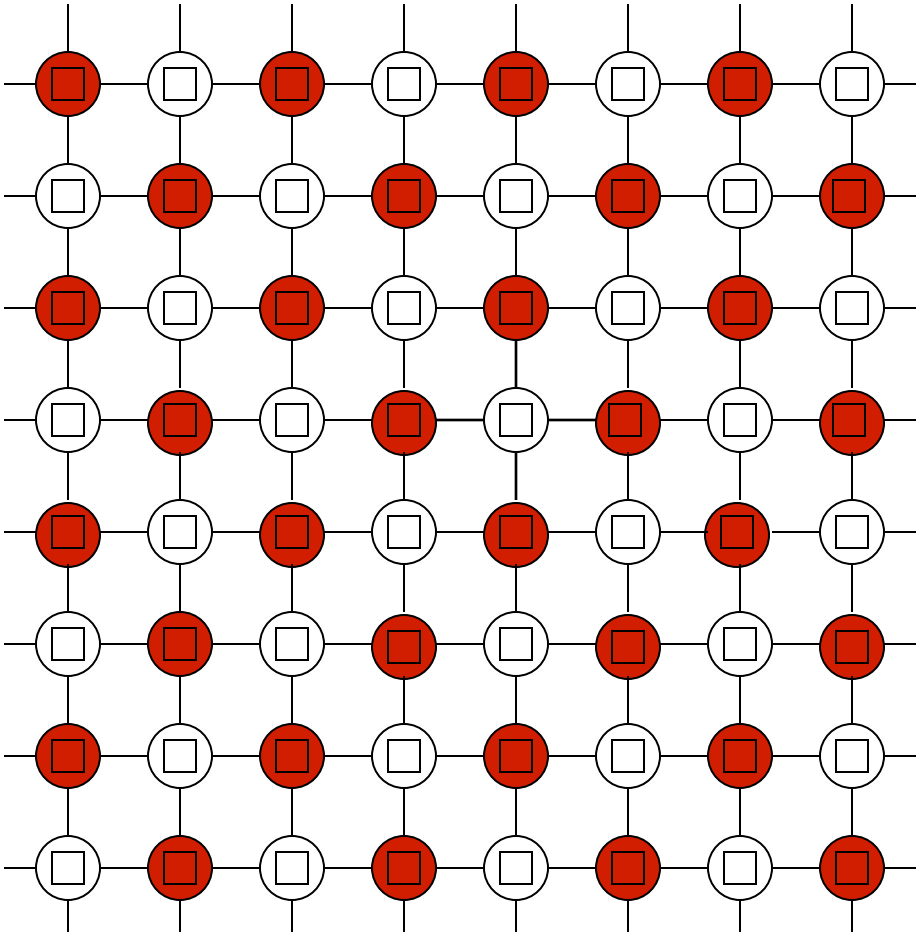
Dependencia y concurrencia para el método Gauss-Seidel



Descomposición

- Método Gauss-Seidel:
  - Conocimiento de la aplicación.
  - Cálculo de cada iteración en dos fases: rojo-blanco.
  - Concurrentemente (N x N) ÷ 2 operaciones.

Descomposición Rojo-Blanco



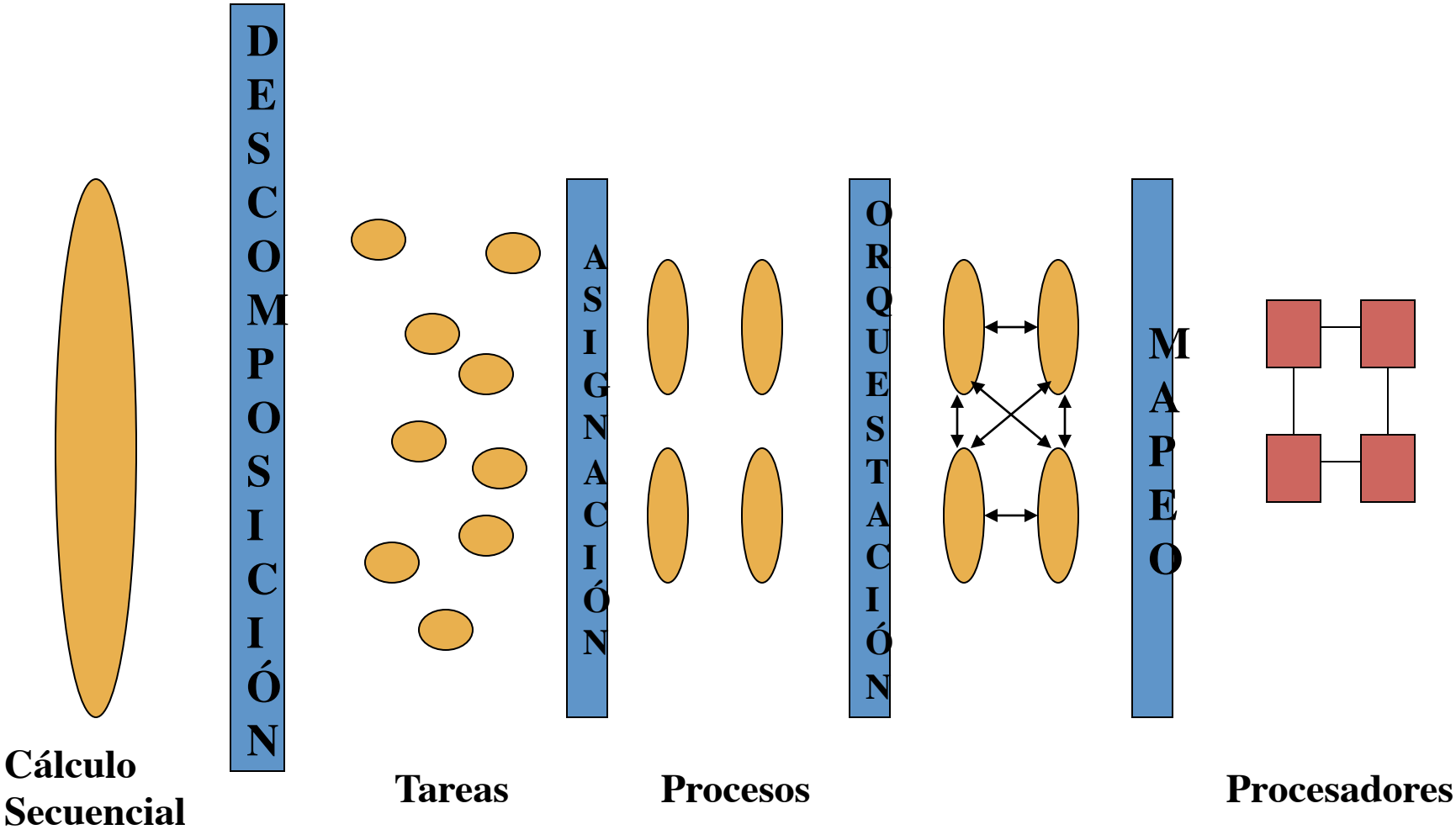
Descomposición

- Uso de método Jacobi:
  - Más caro que el método Gauss-Seidel cuando se trabaja secuencialmente.
  - Permite que el bucle interno pueda ser realizado concurrentemente y asíncronamente.
  - Permite tener *n* tareas independientes.

Descomposición

```
15. while (!done) do                                  /*a sequential loop*/
16.     diff = 0;
17.     for_all i ← 1 to n do                         /*a parallel loop nest*/
18.         for_all j ← 1 to n do
19.             temp = A[i,j];
20.             A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                 A[i,j+1] + A[i+1,j]);
22.             diff += abs(A[i,j] - temp);
23.         end for_all
24.     end for_all
25.     if (diff/(n*n) < TOL) then done = 1;

26. end while
```

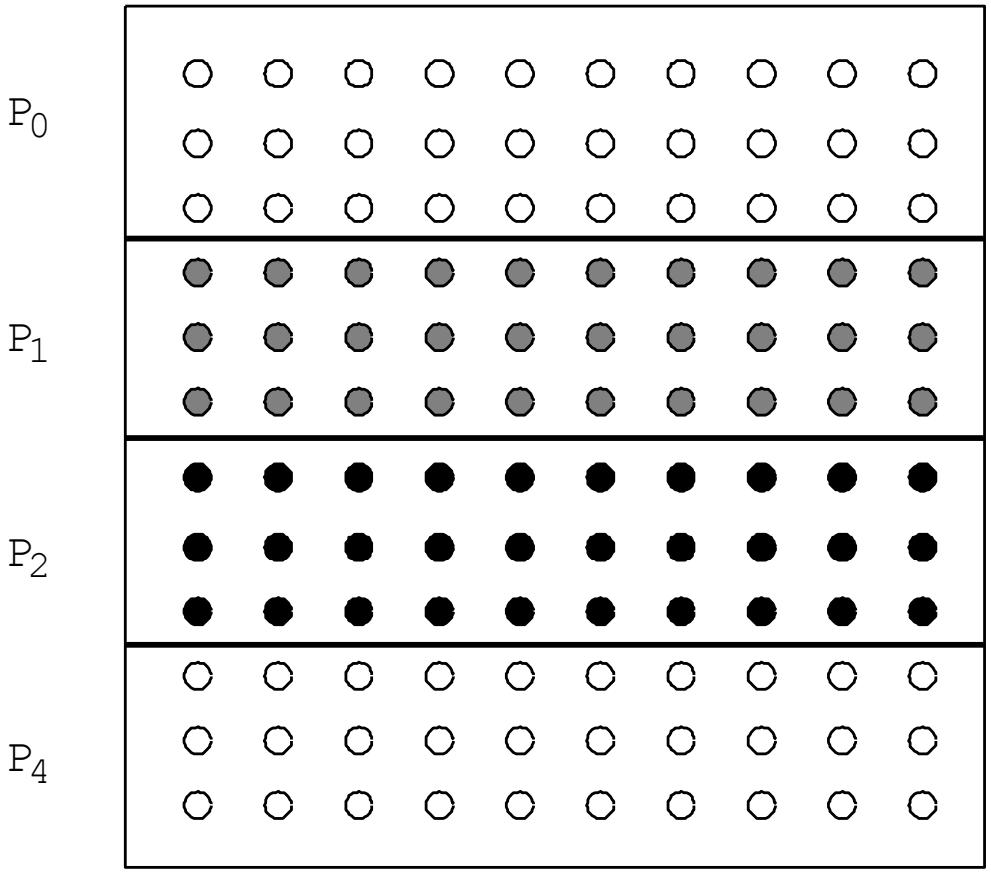
4 etapas en el proceso de paralelización [Culler99]



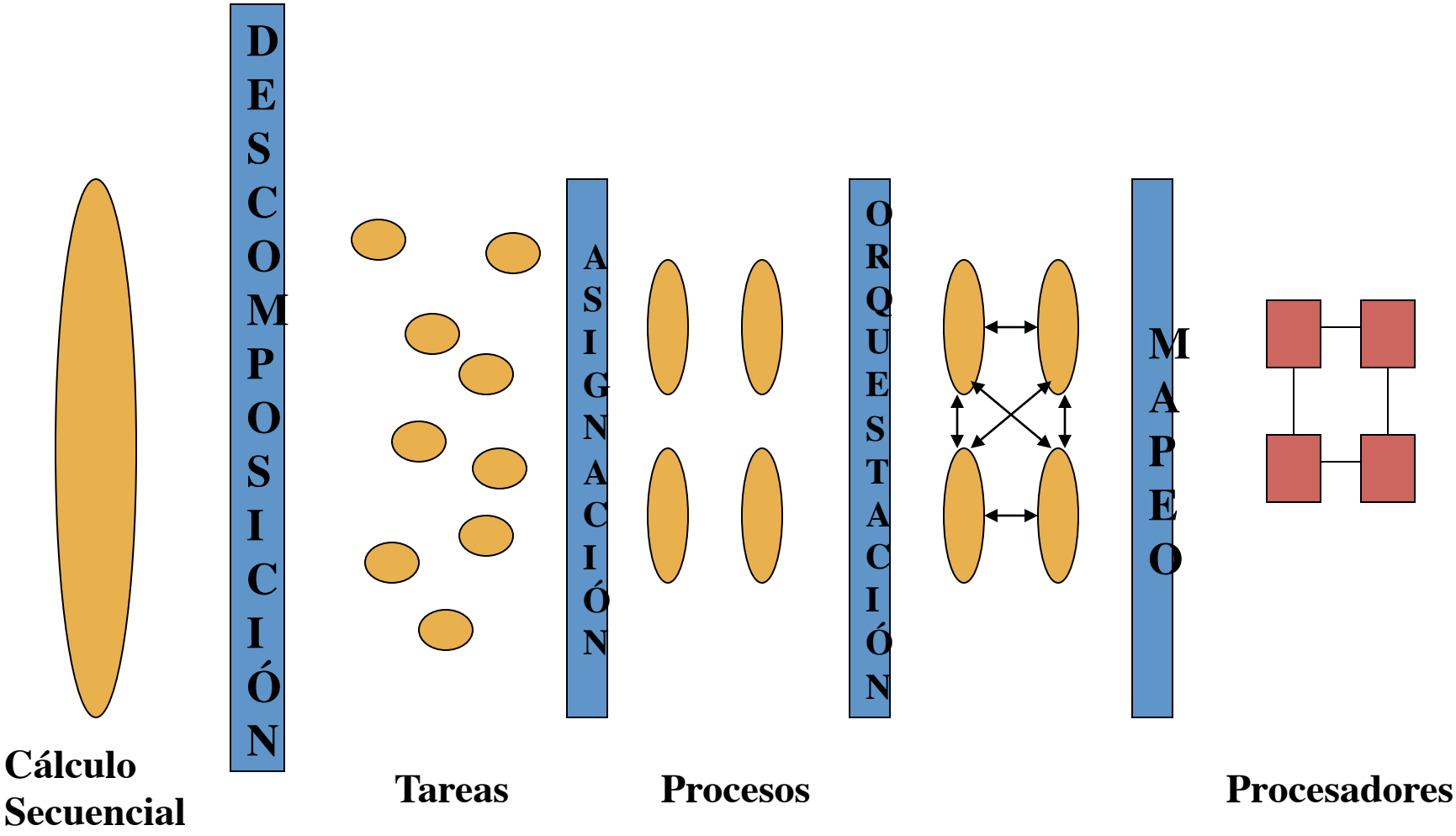
Asignación

- Asignación estática: Descomposición en filas
  - Asignación por bloques de las filas: Fila i asignada al proceso: $\left\lfloor \frac{i}{p} \right\rfloor$
  - Asignación cíclica de las filas a los procesos: al proceso i le son asignadas las filas i, $i+p$, ...
- Asignación dinámica:
  - Tomar el índice de una fila, trabajar sobre la fila, tomar una nueva fila, ...

Asignación



4 etapas en el proceso de paralelización [Culler99]



Orquestación

Requerimientos:

- Comunicación.

  Los valores de los vecinos deben estar disponibles.

- Sincronización.

  Antes de pasar a la próxima iteración, todos los puntos de ésta deben estar evaluados.

Orquestación

Modelos de programación posibles:

- Paralelismo de Datos.

  Ejecución concurrente de bucles, descomposición de datos y cómputo, operadores colectivos.

- Memoria Compartida.

  Creación de procesos, exclusión mutua.

- Pase de Mensajes.

  Creación de procesos, *send/receive* síncronos o asíncronos.

Orquestación
Modelo Paralelismo de Datos

- Asignación dinámica de datos compartidos:
  - G_MALLOC (global malloc)
- Bucles concurrentes:
  - *for_all*

Orquestación
Modelo Paralelismo de Datos

- Descomposición de datos y cálculo:
  - DECOMP. Especifica la asignación de iteraciones a los procesos. Ejemplos:
    - [BLOCK,*,*nprocs*] las filas se particionan entre bloques contiguos entre los *nprocs*.
    - [CYCLIC,*,*nprocs*] las filas se particionan cíclicamente entre los nprocs.
- Operaciones colectivas:
  - REDUCE, BROADCAST, etc.
  - Operaciones *all-to-all*, implementadas de manera eficiente en el sistema base.

```
1.    int n, nprocs;                                  /*grid size (n + 2-by-n + 2) and number of processes*/
2.    float **A, diff = 0;

3.    main()
4.    begin
5.         read(n); read(nprocs);              ;     /*read input grid size and number of processes*/
6.         A ← G_MALLOC (a 2-d array of size n+2 by n+2 doubles);
7.         initialize(A);                           /*initialize the matrix A somehow*/
8.         Solve (A);                               /*call the routine to solve equation*/
9.    end main

10.   procedure Solve(A)                            /*solve the equation system*/
11.        float **A;                               /*A is an (n + 2-by-n + 2) array*/
12.   begin
13.        int i, j, done = 0;
14.        float mydiff = 0, temp;
14a.       DECOMP A[BLOCK,*, nprocs];
15.        while (!done) do                         /*outermost loop over sweeps*/
16.             mydiff = 0;                         /*initialize maximum difference to 0*/
17.             for_all i ← 1 to n do               /*sweep over non-border points of grid*/
18.                  for_all j ← 1 to n do
19.                       temp = A[i,j];           /*save old value of element*/
20.                       A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                            A[i,j+1] + A[i+1,j]);    /*compute average*/
22.                       mydiff[i,j] = abs(A[i,j] - temp);
23.                  end for_all
24.             end for_all
24a.            REDUCE (mydiff, diff, ADD);
25.             if (diff/(n*n) < TOL) then done = 1;
26.        end while
27.   end procedure
```

Orquestación
Modelo Memoria Compartida

- Creación y Terminación de Procesos.
  - CREATE(p, proc, args)
  - WAIT_FOR_END(number)
- Asignación dinámica de datos compartidos.
  - G_MALLOC
- Exclusión Mutua.
  - LOCK(name): adquiere acceso exclusivo
  - UNLOCK(name): libera acceso

Orquestación
Modelo Memoria Compartida

- Sincronización Global:
  - BARRIER(name,number).
  - Cuando *number* número de procesos llegan a la barrera, los procesos pueden continuar con la ejecución.

- Sincronización punto a punto:
  - WAIT(bandera): esperar a que la bandera se active.
  - SIGNAL(bandera): activa la bandera.

```
10.  procedure Solve(A)
11.       float **A;                          /*A is entire n+2-by-n+2 shared array,
                                               as in the sequential program*/
12.  begin
13.       int i,j, pid, done = 0;
14.       float temp, mydiff = 0;             /*private variables*/
14a.      int mymin = 1 + (pid * n/nprocs);   /*assume that n is exactly divisible by*/
14b.      int mymax = mymin + n/nprocs - 1    /*nprocs for simplicity here*/

15.       while (!done) do                    /*outer loop sweeps*/
16.         mydiff = diff = 0;                /*set global diff to 0 (okay for all to do it)*/
16a.        BARRIER(bar1, nprocs);            /*ensure all reach here before anyone modifies diff*/
17.         for i ← mymin to mymax do         /*for each of my rows*/
18.           for j ← 1 to n do               /*for all nonborder elements in that row*/
19.             temp = A[i,j];
20.             A[i,j] = 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
21.                             A[i,j+1] + A[i+1,j]);
22.             mydiff += abs(A[i,j] - temp);
23.           endfor
24.         endfor
25a.        LOCK(diff_lock);                          /*update global diff if necessary*/
25b.          diff += mydiff;
25c.        UNLOCK(diff_lock);
25d.        BARRIER(bar1, nprocs);          /*ensure all reach here before checking if done*/
25e.        if (diff/(n*n) < TOL) then done = 1;  /*check convergence; all get
                                                    same answer*/
25f.        BARRIER(bar1, nprocs);
26.       endwhile
27.  end procedure
```

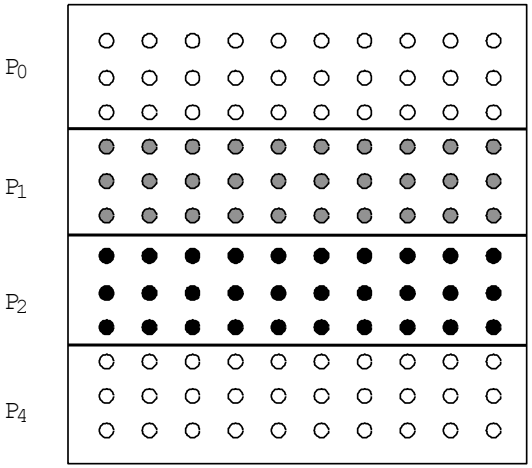
Orquestación
Modelo Paso de Mensajes

- Creación y Terminación de Procesos:
  - CREATE
  - WAIT_FOR_END

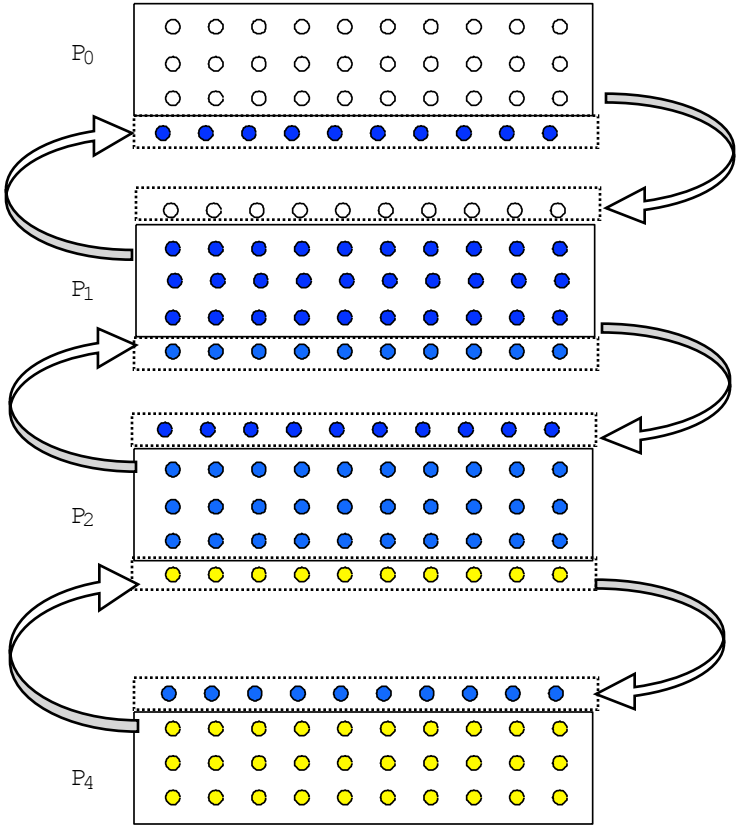
- Comunicación: Transferencia de Datos + Sincronización:
  - SEND(src_addr, size, dest, tag)
  - RECEIVE(buffer_addr, size, src, tag)
  - SEND_ASYNC, RECEIVE_ASYNC

- Sincronización Global:
  - BARRIER

Orquestación
Modelo Paso de Mensajes



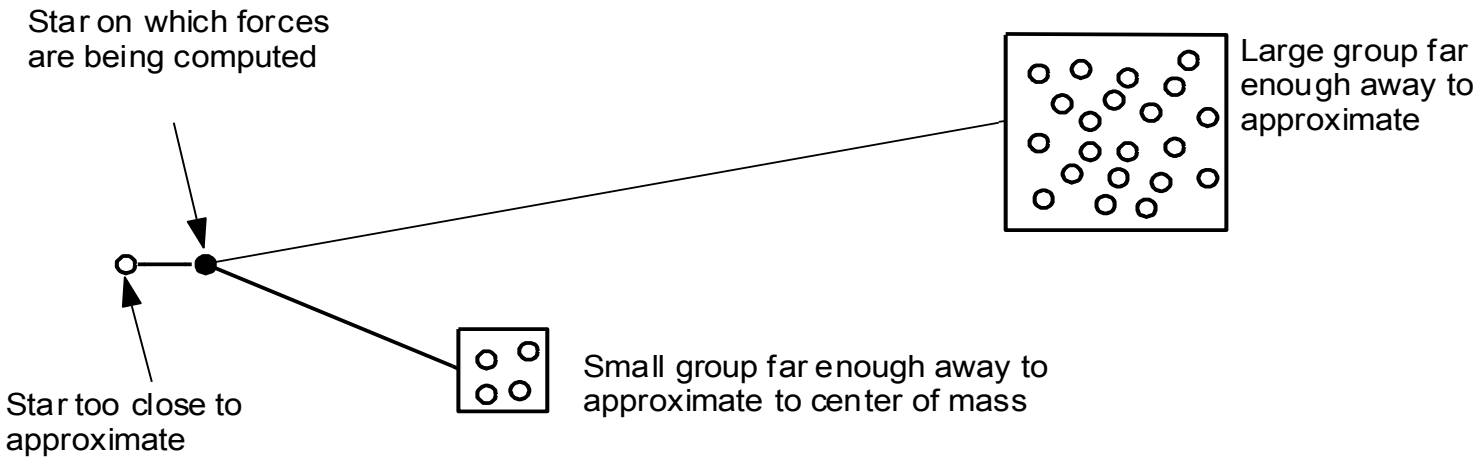
Se añaden filas extras para utilizar Datos de otros procesos



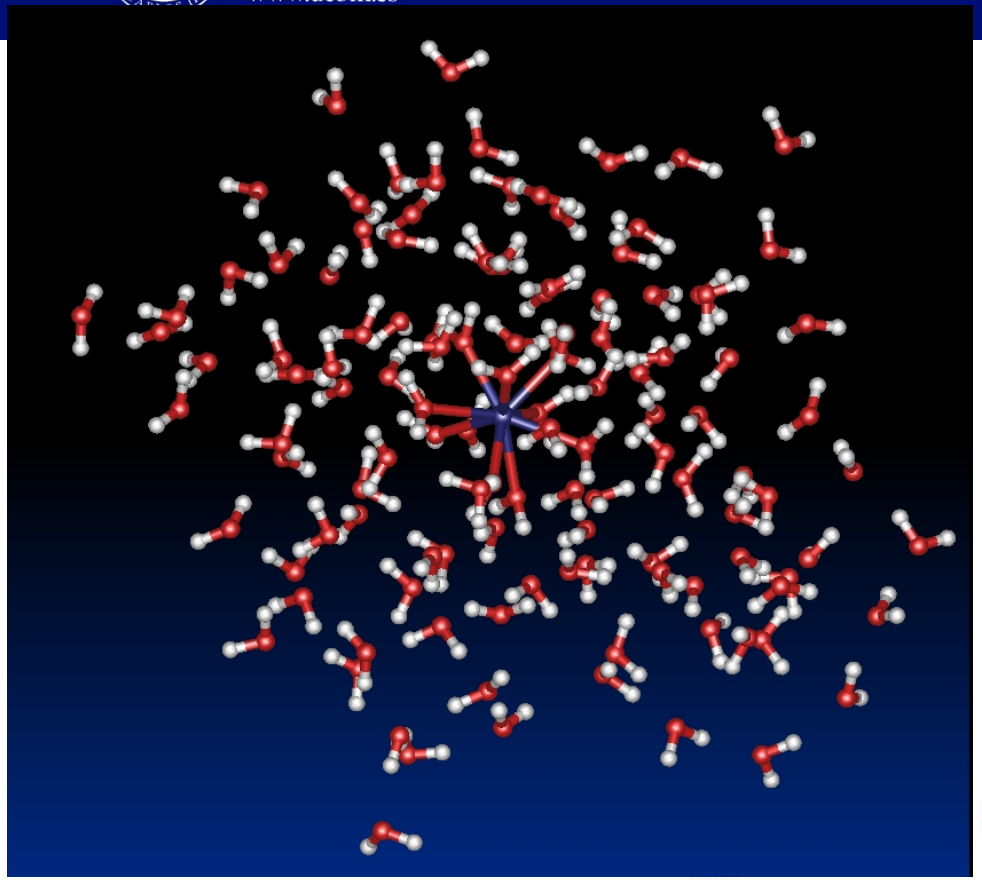
```
10.  procedure Solve()
11.  begin
13.     int i,j, pid, n' = n/nprocs, done = 0;
14.     float temp, tempdiff, mydiff = 0;        /*private variables*/
6.    myA ← malloc(a 2-d array of size [n/nprocs + 2] by n+2);
                                                 /*initialize my rows of A, in an unspecified way*/

15.  while (!done) do
16.     mydiff = 0;                              /*set local diff to 0*/
      /* Exchange border rows of neighbors into myA[0,*] and myA[n'+1,*]*/
16a.    if (pid != 0) then SEND(&myA[1,0],n*sizeof(float),pid-1,ROW);
16b.    if (pid != nprocs-1) then
          SEND(&myA[n',0],n*sizeof(float),pid+1,ROW);
16c.    if (pid != 0) then RECEIVE(&myA[0,0],n*sizeof(float),pid-1,ROW);
16d.    if (pid != nprocs-1) then
          RECEIVE(&myA[n'+1,0],n*sizeof(float), pid+1,ROW);
17.     for i ← 1 to n' do                       /*for each of my (nonghost) rows*/
18.       for j ← 1 to n do                      /*for all nonborder elements in that row*/
19.         temp = myA[i,j];
20.         myA[i,j] = 0.2 * (myA[i,j] + myA[i,j-1] + myA[i-1,j] +
21.           myA[i,j+1] + myA[i+1,j]);
22.         mydiff += abs(myA[i,j] - temp);
23.       endfor
24.     endfor
                                                 /*communicate local diff values and determine if
                                                 done; can be replaced by reduction and broadcast*/
25a.      if (pid != 0) then                     /*process 0 holds global total diff*/
25b.        SEND(mydiff,sizeof(float),0,DIFF);
25c.        RECEIVE(done,sizeof(int),0,DONE);
25d.      else                                   /*pid 0 does this*/
25e.        for i ← 1 to nprocs-1 do             /*for each other process*/
25f.          RECEIVE(tempdiff,sizeof(float),*,DIFF);
25g.        mydiff += tempdiff;                  /*accumulate into total*/
25h.        endfor
25i         if (mydiff/(n*n) < TOL) then  done = 1;
25j.        for i ← 1 to nprocs-1 do             /*for each other process*/
25k.          SEND(done,sizeof(int),i,DONE);
25l.        endfor
25m.    endif
26.  endwhile
27.  end procedure
```

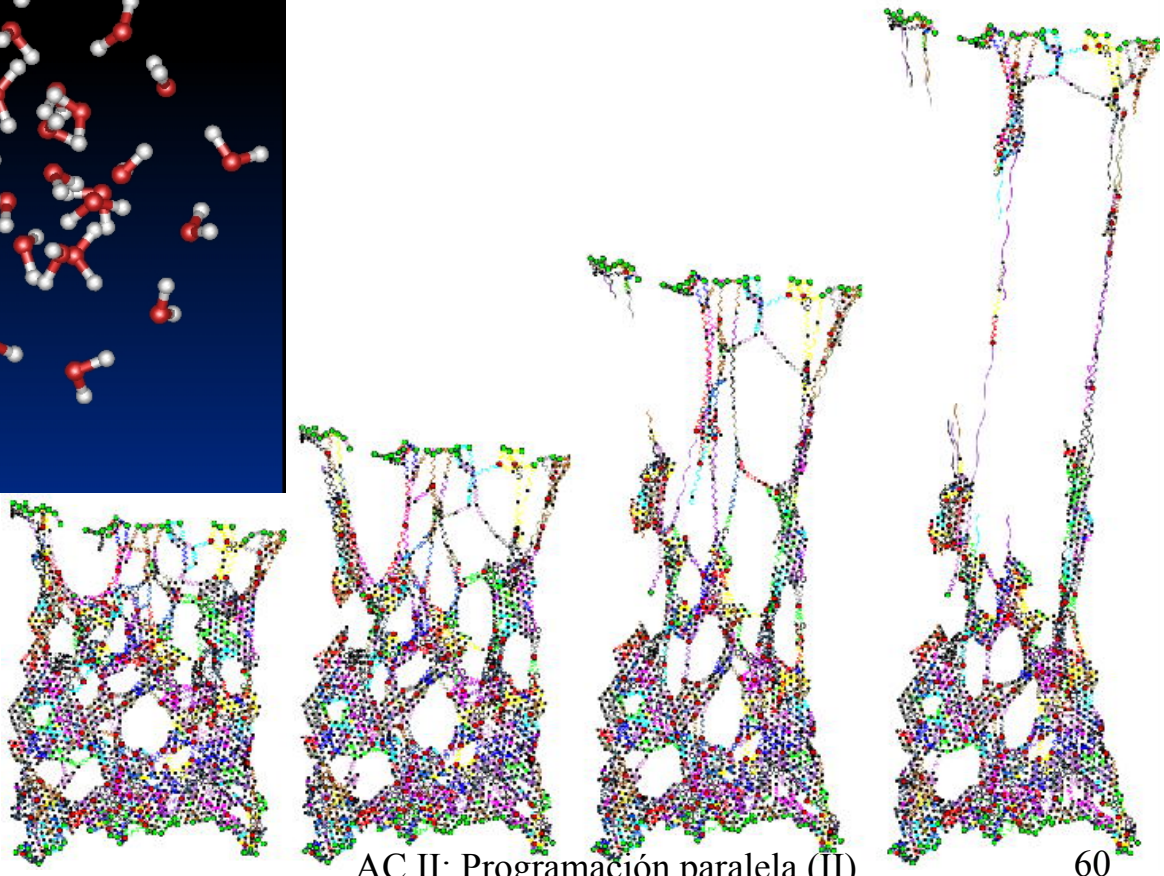
Simulación de la Evolución de las Galaxias

- Simulación de las interacciones de las estrellas en el tiempo.
- El cálculo de fuerzas es caro.
  - $O(n^2)$
  - Uso de métodos jerárquicos para minimizar costes O(n log n)



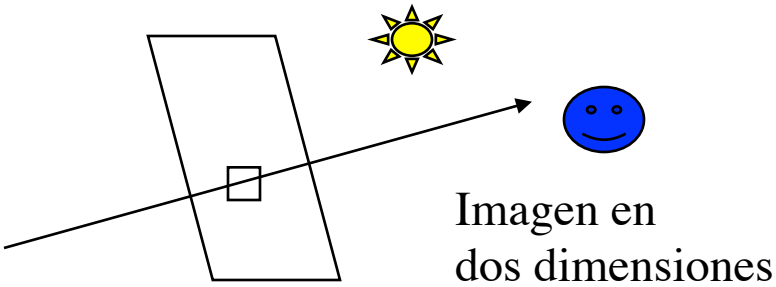
Simulación dinámica molecular





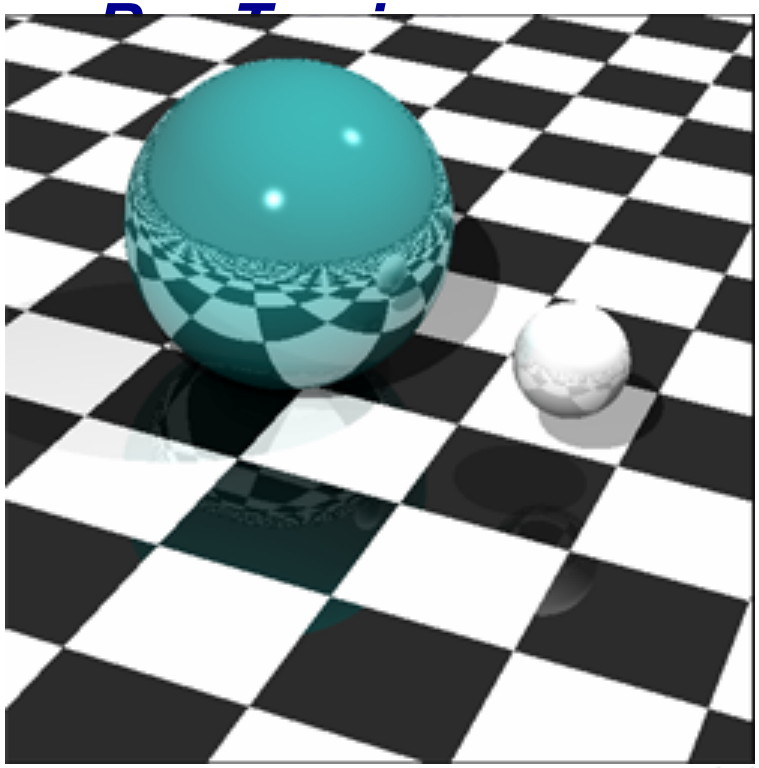
Mostrar escenas utilizando *Ray Tracing*

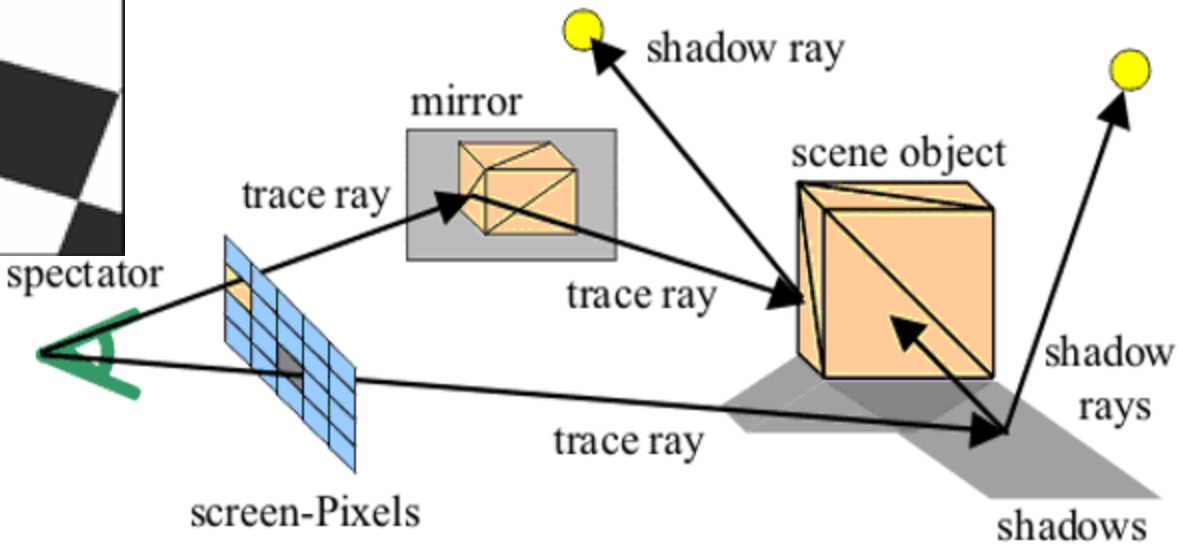
- Transformación 3D a 2D exige:
  - Cálculo de valores de color, opacidad, brillo
    - Simulación de cómo llegan rayos de luz sobre cada pixel
- Paralelismo ligado a los rayos de luz



Escena en tres dimensiones

Mostrar escenas utilizando





Bibliografía

D. Culler, J. Pal Singh, A. Gupta. *Parallel Computer Architecture. A Hardware/Software Approach*. Morgan Kaufmann. Capítulo 2.