

# **Algoritmos Paralelos**

**(clase 22.09.15)**

Prof. J.Fiestas

# Solucion de Ecuaciones Diferenciales Parciales

Expresan cambios en funciones continuas en modelos de sistemas físicos y su evolución. Se encuentran en todas las ciencias, ingeniería e industria.

$$\frac{\partial u}{\partial x}(x, y) = 0.$$

Con la solución

$$u(x, y) = f(y),$$

e.g. ecuaciones de calor, de difusión, de onda

# Solucion de Ecuaciones Diferenciales Parciales

Aplicación de diferencias finitas en cálculo de derivadas.

$$\frac{df}{dx} = \frac{f(x_0 + \delta) - f(x_0 - \delta)}{2\delta}$$

$$\frac{d^2f}{dx^2} = \frac{f(x_0 + \delta) - 2f(x_0) + f(x_0 - \delta)}{\delta^2}$$

Mas exacto, segun la serie de Taylor evaluada en  $x_0$

$$f(a) + \frac{f'(a)}{1!}(x - a) + \frac{f''(a)}{2!}(x - a)^2 + \frac{f'''(a)}{3!}(x - a)^3 + \dots$$

# Solucion de Ecuaciones Diferenciales Parciales

EDP de la forma

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Du_x + Eu_y + F = 0$$

cuyas soluciones satisfacen la condición:

1.  $B^2 - AC < 0$  , son llamadas elípticas (Laplace)
2.  $B^2 - AC = 0$  , son llamadas parabólicas  
(ecuaciones de calor)
3.  $B^2 - AC > 0$  , son llamadas hiperbólicas  
(ecuación de onda)

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

Es un modelo matemático de flujo de calor y viscozidad.

u es la temperatura, x-y las coordenadas espaciales

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

En un grid regular (i.e.  $dx$ ,  $dy$  constantes), se puede substituir la **ecuación finita diferencial** de primer orden para una derivada parcial

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} + O(\Delta x)$$

$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1} - u_{i,j}}{\Delta y} + O(\Delta y)$$

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

En un grid regular (i.e.  $dx$ ,  $dy$  constantes), se puede substituir la **ecuación finita diferencial** de segundo orden para una derivada parcial

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{(\Delta x)^2} + O((\Delta x)^2)$$

$$\frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} + O((\Delta y)^2)$$

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

Substituyendo en la ecuación de Laplace, tenemos

$$\frac{u_{i+1,j}^n - 2u_{i,j}^{n+1} + u_{i-1,j}^n}{(\Delta x)^2} + \frac{u_{i,j+1}^n - 2u_{i,j}^{n+1} + u_{i,j-1}^n}{(\Delta y)^2} = O((\Delta x)^2, (\Delta y)^2)$$

donde se puede descartar el término de error y asumir que  $\Delta x$  y  $\Delta y$  tienen la misma longitud

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

Al resolver la ecuación para  $u_{i,j}^{n+1}$  y substrayendo  $u_{i,j}^n$  de ambos lados, se obtiene

$$\Delta u_{i,j}^{n+1} = \frac{u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n}{4} - u_{i,j}^n$$

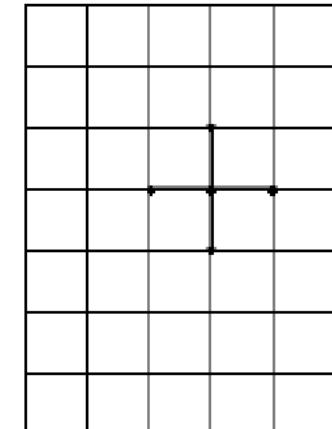
$$u_{i,j}^{n+1} = u_{i,j}^n + \Delta u_{i,j}^{n+1}$$

## Método de las diferencias finitas

# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Laplace (2D)

Al resolver la ecuación para  $u_{i,j}^{n+1}$  y substrayendo  $u_{i,j}^n$  de ambos lados, se obtiene

$$u(i, j) = 0.25 * ( uold(i - 1, j) + uold(i + 1, j) + uold(i, j - 1) + uold(i, j + 1) )$$



# Ecuación de Laplace (2D): condiciones de frontera

Inicialmente, las condiciones de frontera son

$$0 \leq x \leq L$$

$$0 \leq y \leq L$$

$$u(0, y) = 0$$

$$u(L, y) = u_{\max} \sin\left(\pi \frac{y}{L}\right)$$

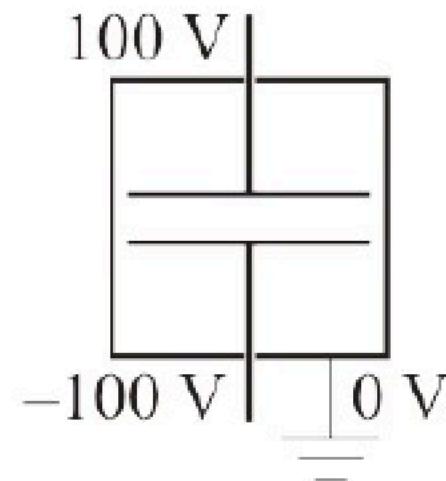
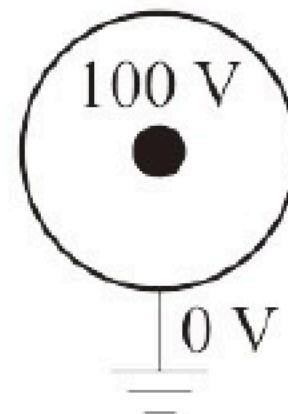
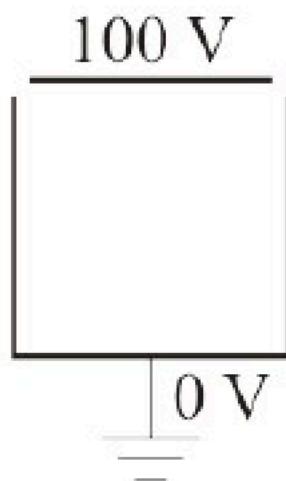
$$u(x, 0) = 0$$

$$u(x, L) = 0$$

$$u_{i,j}^0 = 0$$

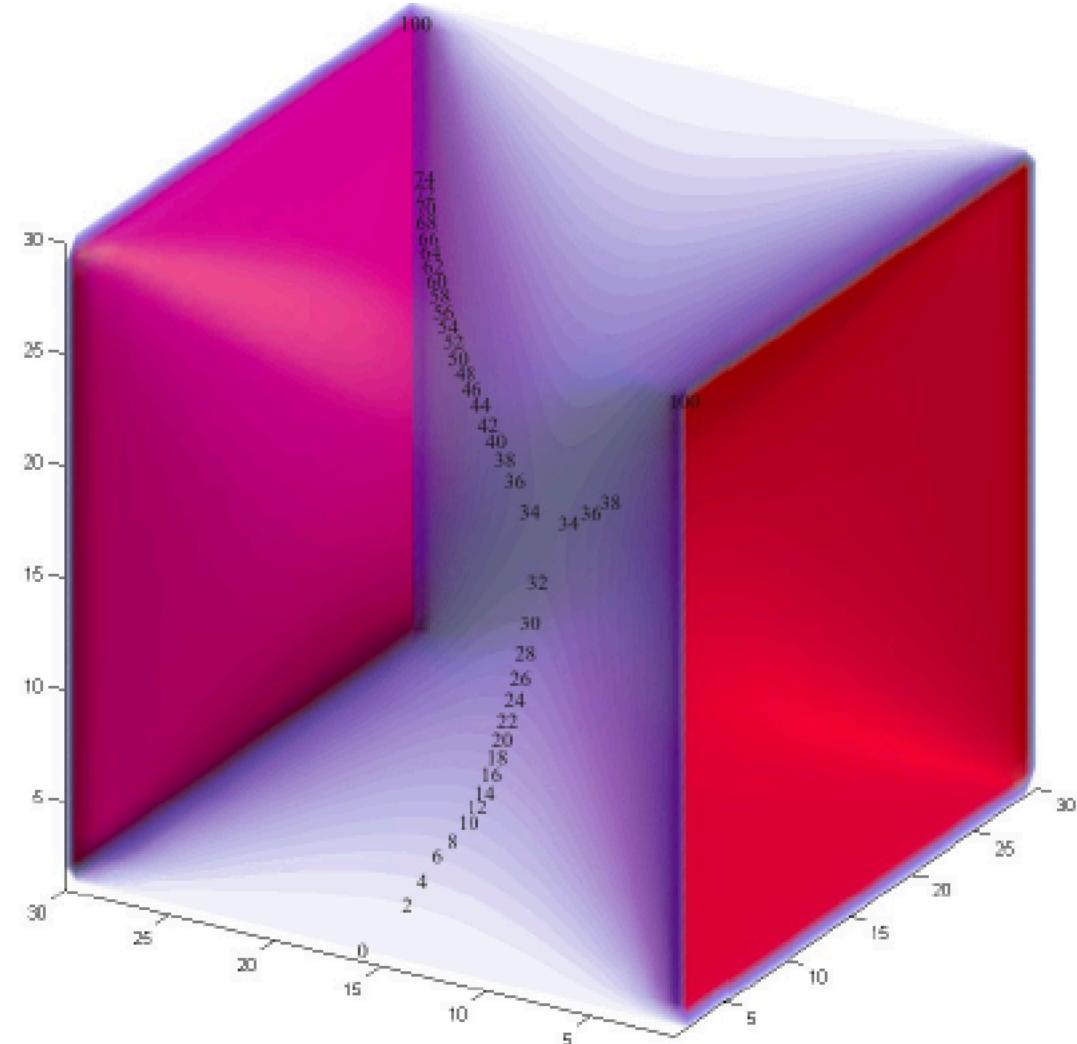
# Ecuación de Laplace (2D): condiciones de frontera

Pueden ser dados por voltajes (pueden ser internos o externos)



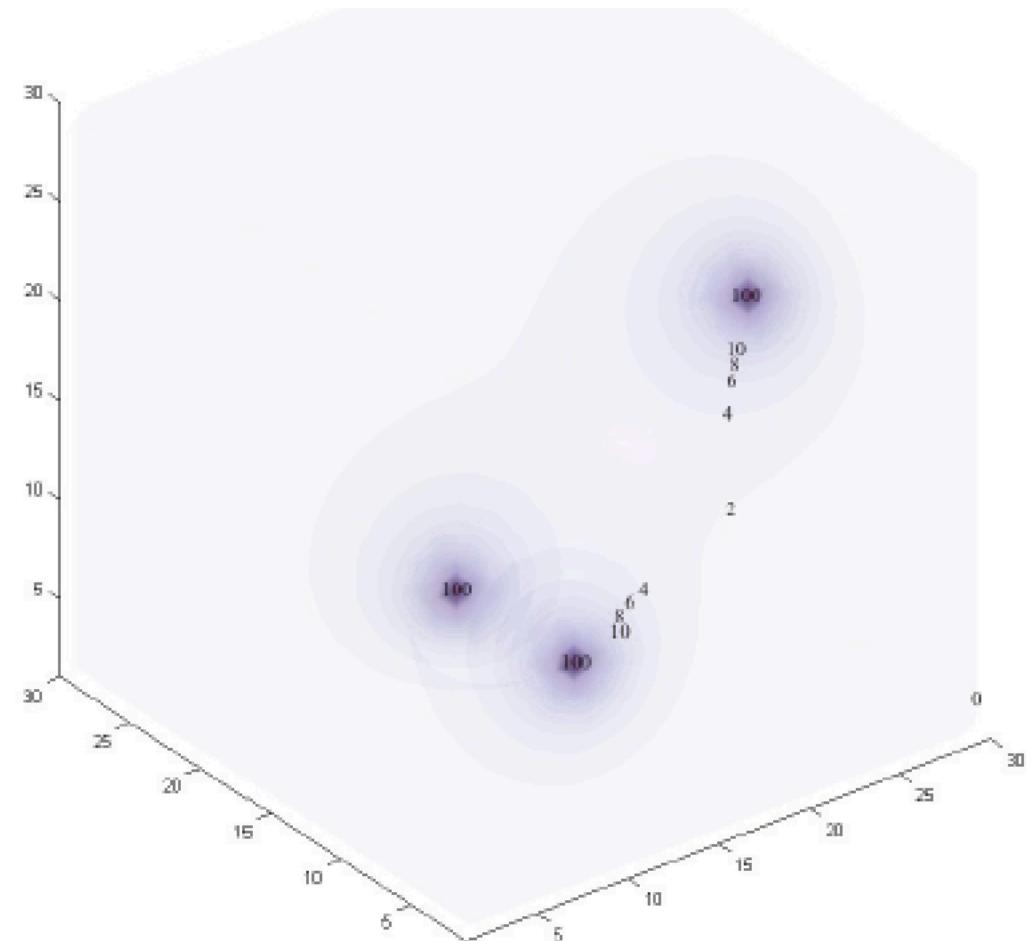
# Ecuación de Laplace (2D): condiciones de frontera

Temperaturas de frontera en 3D:  
100° en lados opuestos y  
0° en los otros dos



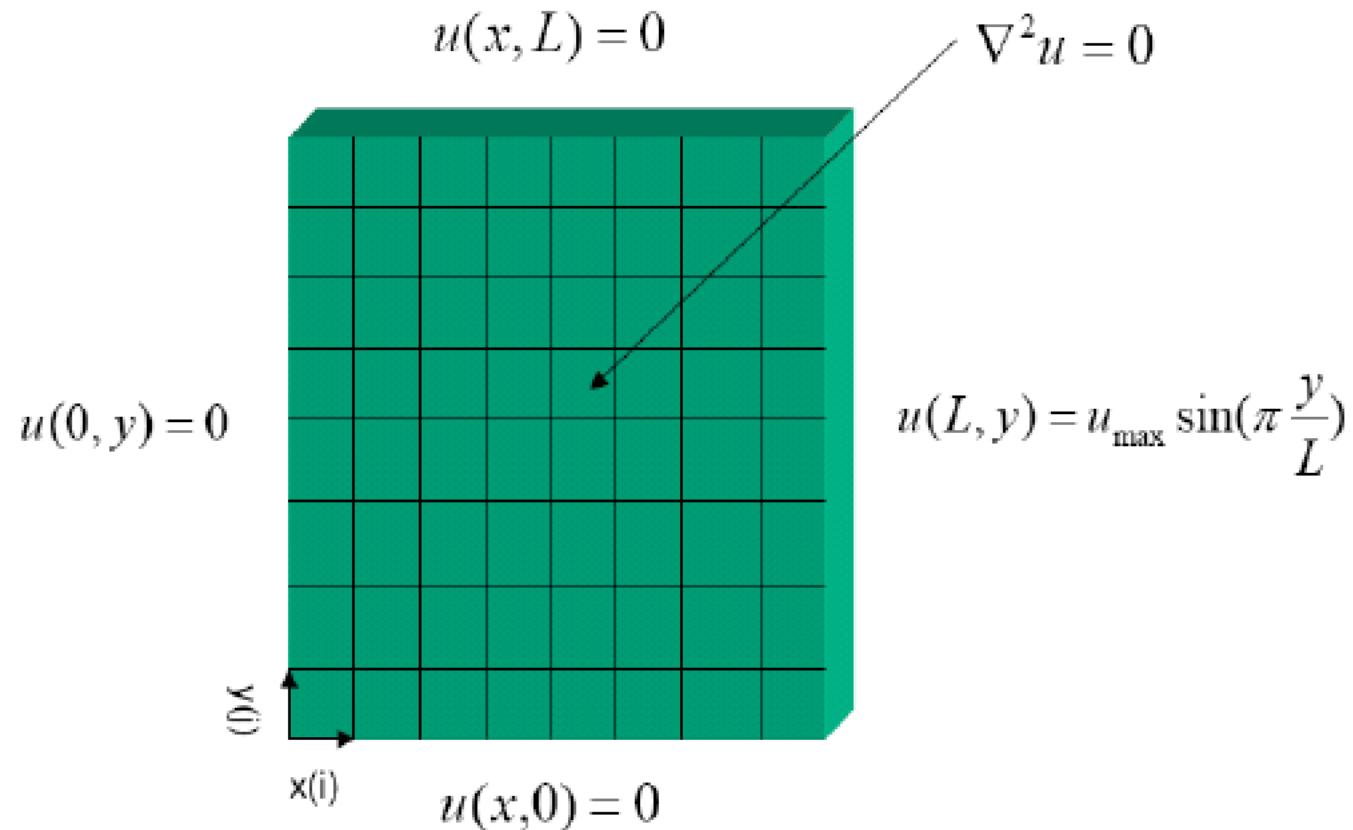
# Ecuación de Laplace (2D): condiciones de frontera

Temperaturas  
de frontera en 3D:  
3 fuentes de calor  
con paredes de  
 $0^\circ$



# Ecuación de Laplace (2D):

Es decir, dominio, condiciones de frontera e iniciales



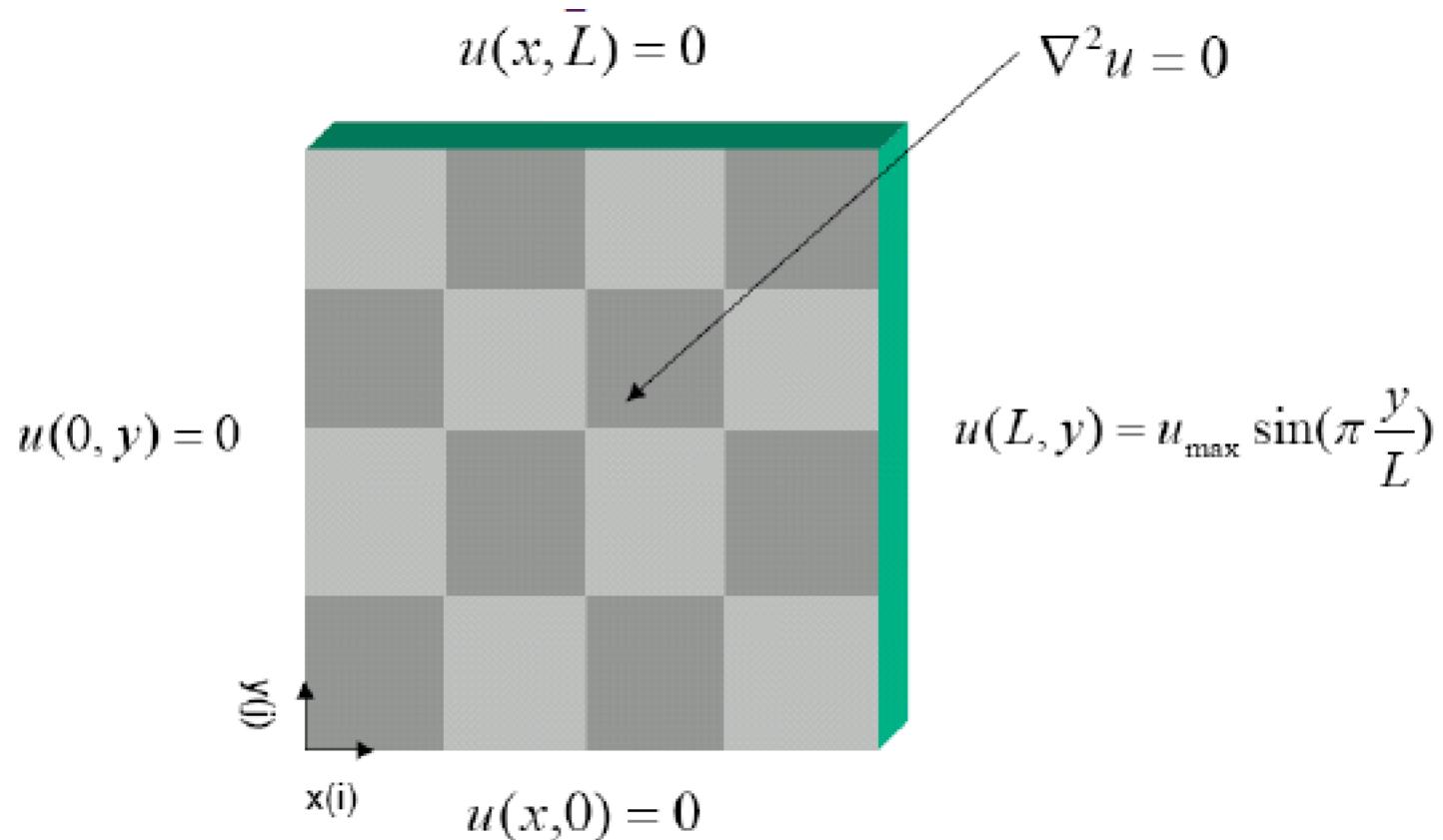
# Ecuación de Laplace (2D):

Código en serie

- `for (i=1; i <= NR; i++)`
- `for (j=1; j <= NC; j++)`
- `u[i][j] = 0.25*( uold[i+1][j] +`
- `uold[i-1][j] +`
- `uold[i][j+1] +`
- `uold[i][j-1] );`

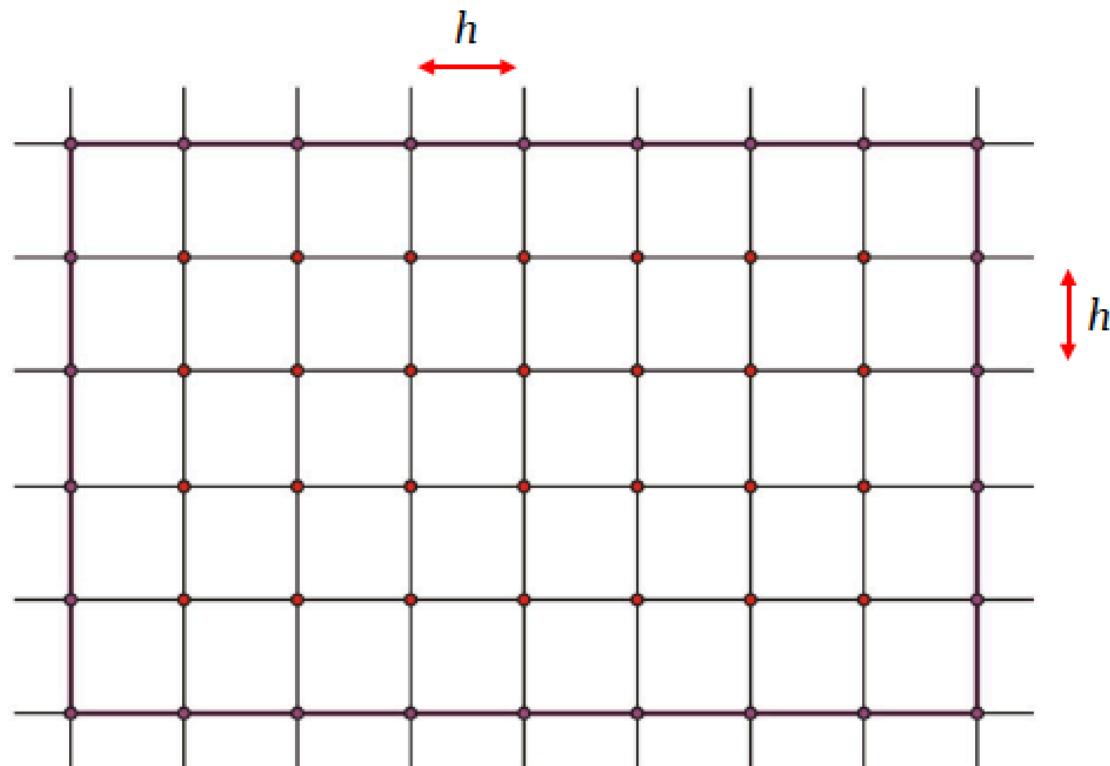
# Ecuación de Laplace (2D):

Descomposición de dominio



# Ecuación de Laplace (2D):

Se construye una malla de  $n_x \times n_y$  puntos y distancia constante  $h$



# Ecuación de Laplace (2D):

Representación en matriz  $u_{ij}$

$$\begin{matrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} & u_{1,5} & u_{1,6} & u_{1,7} & u_{1,8} & u_{1,9} \\ u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} & u_{2,5} & u_{2,6} & u_{2,7} & u_{2,8} & u_{2,9} \\ u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} & u_{3,5} & u_{3,6} & u_{3,7} & u_{3,8} & u_{3,9} \\ u_{4,1} & u_{4,2} & u_{4,3} & u_{4,4} & u_{4,5} & u_{4,6} & u_{4,7} & u_{4,8} & u_{4,9} \\ u_{5,1} & u_{5,2} & u_{5,3} & u_{5,4} & u_{5,5} & u_{5,6} & u_{5,7} & u_{5,8} & u_{5,9} \\ u_{6,1} & u_{6,2} & u_{6,3} & u_{6,4} & u_{6,5} & u_{6,6} & u_{6,7} & u_{6,8} & u_{6,9} \end{matrix}$$

# Ecuación de Laplace (2D):

Se conocen las condiciones de frontera

$$u_{1,1}^* \ u_{1,2}^* \ u_{1,3}^* \ u_{1,4}^* \ u_{1,5}^* \ u_{1,6}^* \ u_{1,7}^* \ u_{1,8}^* \ u_{1,9}^*$$

$$u_{2,1}^* \qquad \qquad \qquad u_{2,9}^*$$

$$u_{3,1}^* \qquad \qquad \qquad u_{3,9}^*$$

$$u_{4,1}^* \qquad \qquad \qquad u_{4,9}^*$$

$$u_{5,1}^* \qquad \qquad \qquad u_{5,9}^*$$

$$u_{6,1}^* \ u_{6,2}^* \ u_{6,3}^* \ u_{6,4}^* \ u_{6,5}^* \ u_{6,6}^* \ u_{6,7}^* \ u_{6,8}^* \ u_{6,9}^*$$

# Ecuación de Laplace (2D):

Y se debe averiguar los valores intermedios

$$u_{1,1}^{\bullet} \ u_{1,2}^{\bullet} \ u_{1,3}^{\bullet} \ u_{1,4}^{\bullet} \ u_{1,5}^{\bullet} \ u_{1,6}^{\bullet} \ u_{1,7}^{\bullet} \ u_{1,8}^{\bullet} \ u_{1,9}^{\bullet}$$

$$u_{2,1}^{\bullet} \ u_{2,2}^{\bullet} \ u_{2,3}^{\bullet} \ u_{2,4}^{\bullet} \ u_{2,5}^{\bullet} \ u_{2,6}^{\bullet} \ u_{2,7}^{\bullet} \ u_{2,8}^{\bullet} \ u_{2,9}^{\bullet}$$

$$u_{3,1}^{\bullet} \ u_{3,2}^{\bullet} \ u_{3,3}^{\bullet} \ u_{3,4}^{\bullet} \ u_{3,5}^{\bullet} \ u_{3,6}^{\bullet} \ u_{3,7}^{\bullet} \ u_{3,8}^{\bullet} \ u_{3,9}^{\bullet}$$

$$u_{4,1}^{\bullet} \ u_{4,2}^{\bullet} \ u_{4,3}^{\bullet} \ u_{4,4}^{\bullet} \ u_{4,5}^{\bullet} \ u_{4,6}^{\bullet} \ u_{4,7}^{\bullet} \ u_{4,8}^{\bullet} \ u_{4,9}^{\bullet}$$

$$u_{5,1}^{\bullet} \ u_{5,2}^{\bullet} \ u_{5,3}^{\bullet} \ u_{5,4}^{\bullet} \ u_{5,5}^{\bullet} \ u_{5,6}^{\bullet} \ u_{5,7}^{\bullet} \ u_{5,8}^{\bullet} \ u_{5,9}^{\bullet}$$

$$u_{6,1}^{\bullet} \ u_{6,2}^{\bullet} \ u_{6,3}^{\bullet} \ u_{6,4}^{\bullet} \ u_{6,5}^{\bullet} \ u_{6,6}^{\bullet} \ u_{6,7}^{\bullet} \ u_{6,8}^{\bullet} \ u_{6,9}^{\bullet}$$

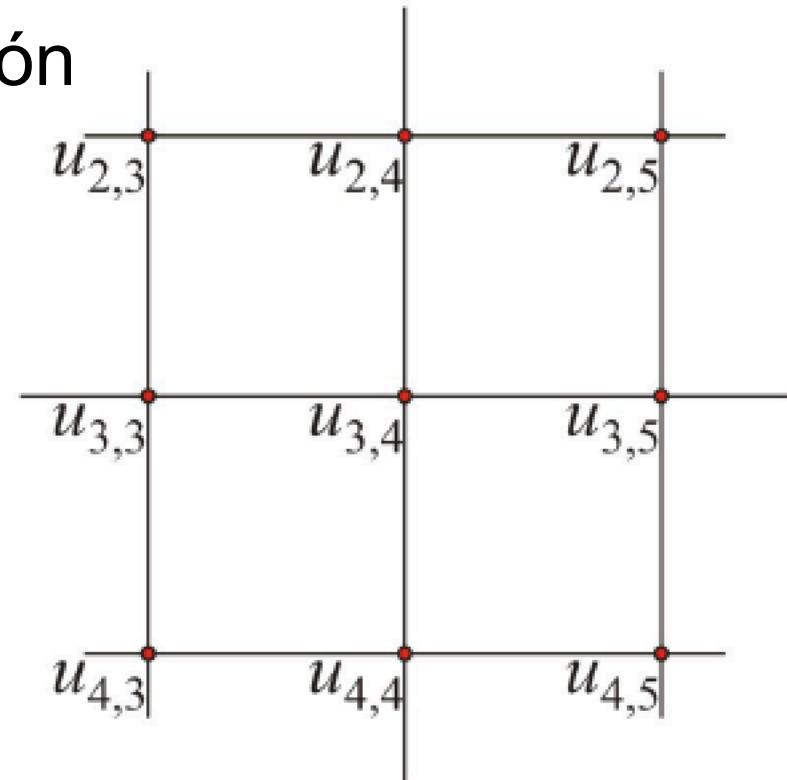
# Ecuación de Laplace (2D):

e.g. calculemos en esta región

$u_{1,1}^*$	$u_{1,2}^*$	$u_{1,3}^*$	$u_{1,4}^*$	$u_{1,5}^*$	$u_{1,6}^*$	$u_{1,7}^*$	$u_{1,8}^*$	$u_{1,9}^*$
$u_{2,1}^*$	$u_{2,2}^*$	$u_{2,3}^*$	$u_{2,4}^*$	$u_{2,5}^*$	$u_{2,6}^*$	$u_{2,7}^*$	$u_{2,8}^*$	$u_{2,9}^*$
$u_{3,1}^*$	$u_{3,2}^*$	$u_{3,3}^*$	$u_{3,4}^*$	$u_{3,5}^*$	$u_{3,6}^*$	$u_{3,7}^*$	$u_{3,8}^*$	$u_{3,9}^*$
$u_{4,1}^*$	$u_{4,2}^*$	$u_{4,3}^*$	$u_{4,4}^*$	$u_{4,5}^*$	$u_{4,6}^*$	$u_{4,7}^*$	$u_{4,8}^*$	$u_{4,9}^*$
$u_{5,1}^*$	$u_{5,2}^*$	$u_{5,3}^*$	$u_{5,4}^*$	$u_{5,5}^*$	$u_{5,6}^*$	$u_{5,7}^*$	$u_{5,8}^*$	$u_{5,9}^*$
$u_{6,1}^*$	$u_{6,2}^*$	$u_{6,3}^*$	$u_{6,4}^*$	$u_{6,5}^*$	$u_{6,6}^*$	$u_{6,7}^*$	$u_{6,8}^*$	$u_{6,9}^*$

# Ecuación de Laplace (2D):

e.g. calculemos en esta región



Y aplicando

$$u(x, y) = \frac{u(x-h, y) + u(x+h, y) + u(x, y-h) + u(x, y+h)}{4}$$

# Ecuación de Laplace (2D):

Con  $x=x_3$ ,  $y=y_4$

$$-4u(x_3, y_4) + u(x_3 - h, y_4) + u(x_3 + h, y_4) + u(x_3, y_4 - h) + u(x_3, y_4 + h) = 0$$

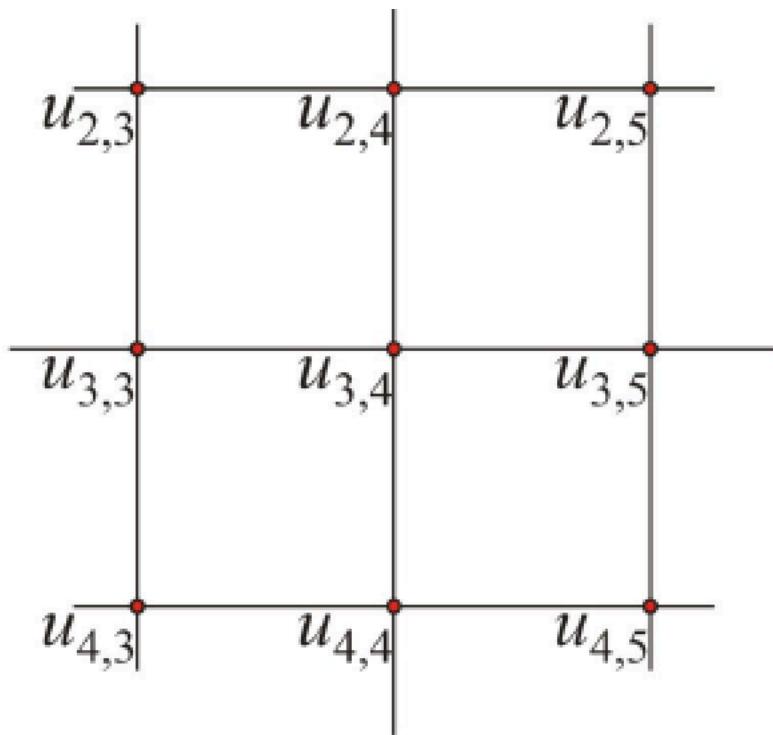
Ademas:

$$x_3 - h = x_2$$

$$x_3 + h = x_4$$

$$y_4 - h = y_3$$

$$y_4 + h = y_5$$

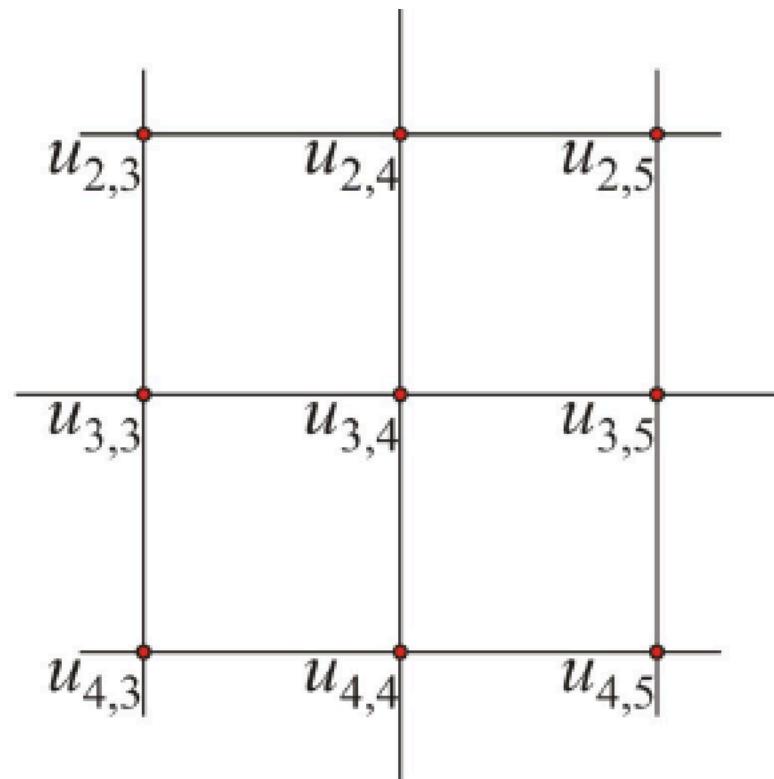


# Ecuación de Laplace (2D):

Simplificando

$$-4u(x_3, y_4) + u(x_2, y_4) + u(x_4, y_4) + u(x_3, y_3) + u(x_3, y_5) = 0$$

$$-4u_{3,4} + u_{2,4} + u_{4,4} + u_{3,3} + u_{3,5} = 0$$



# Ecuación de Laplace (2D):

Si uno de los valores adyacentes esta en la frontera

$$u_{1,1}^* \ u_{1,2}^* \ u_{1,3}^* \ u_{1,4}^* \ u_{1,5}^* \ u_{1,6}^* \ u_{1,7}^* \ u_{1,8}^* \ u_{1,9}^* -4u_{2,4} + u_{1,4} + u_{3,4} + u_{2,3} + u_{2,5} = 0$$

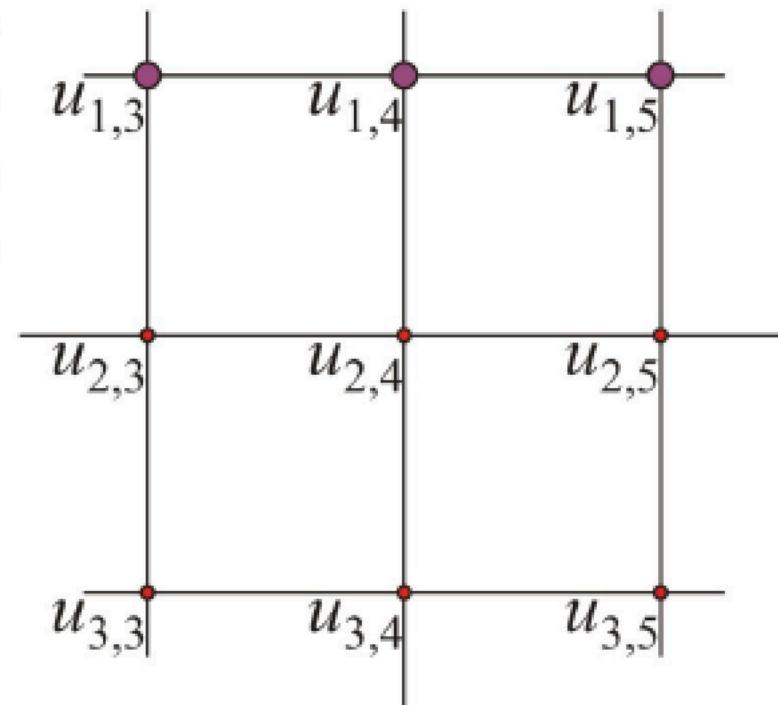
$$u_{2,1}^* \ u_{2,2}^* \ u_{2,3}^* \ u_{2,4}^* \ u_{2,5}^* \ u_{2,6}^* \ u_{2,7}^* \ u_{2,8}^* \ u_{2,9}^*$$

$$u_{3,1}^* \ u_{3,2}^* \ u_{3,3}^* \ u_{3,4}^* \ u_{3,5}^* \ u_{3,6}^* \ u_{3,7}^* \ u_{3,8}^* \ u_{3,9}^*$$

$$u_{4,1}^* \ u_{4,2}^* \ u_{4,3}^* \ u_{4,4}^* \ u_{4,5}^* \ u_{4,6}^* \ u_{4,7}^* \ u_{4,8}^* \ u_{4,9}^*$$

$$u_{5,1}^* \ u_{5,2}^* \ u_{5,3}^* \ u_{5,4}^* \ u_{5,5}^* \ u_{5,6}^* \ u_{5,7}^* \ u_{5,8}^* \ u_{5,9}^*$$

$$u_{6,1}^* \ u_{6,2}^* \ u_{6,3}^* \ u_{6,4}^* \ u_{6,5}^* \ u_{6,6}^* \ u_{6,7}^* \ u_{6,8}^* \ u_{6,9}^*$$

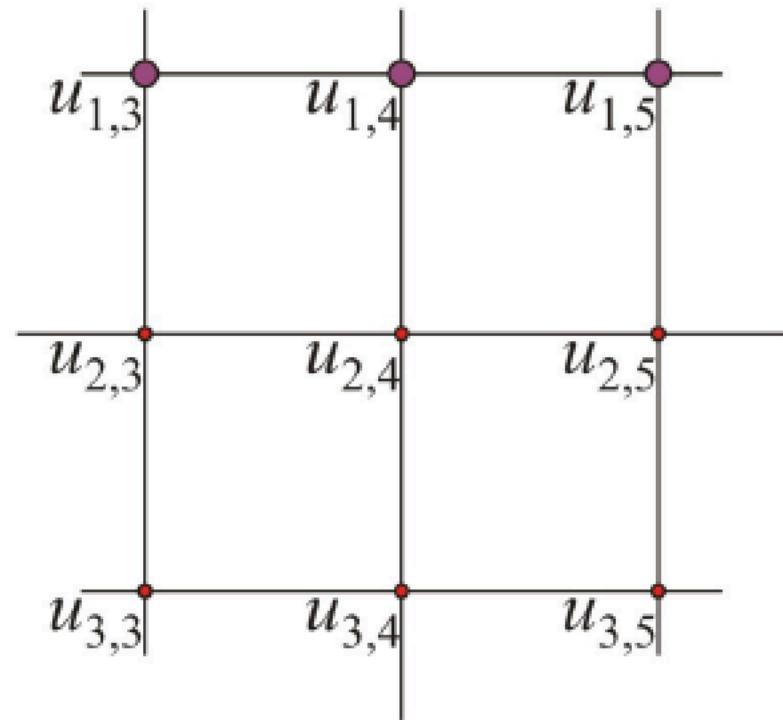


# Ecuación de Laplace (2D):

Es un valor conocido

$$-4u_{2,4} + u_{1,4} + u_{3,4} + u_{2,3} + u_{2,5} = 0$$

$$-4u_{2,4} + u_{3,4} + u_{2,3} + u_{2,5} = -u_{1,4}$$



# Ecuación de Laplace (2D):

Es decir, para cada punto interior se puede escribir una ecuación

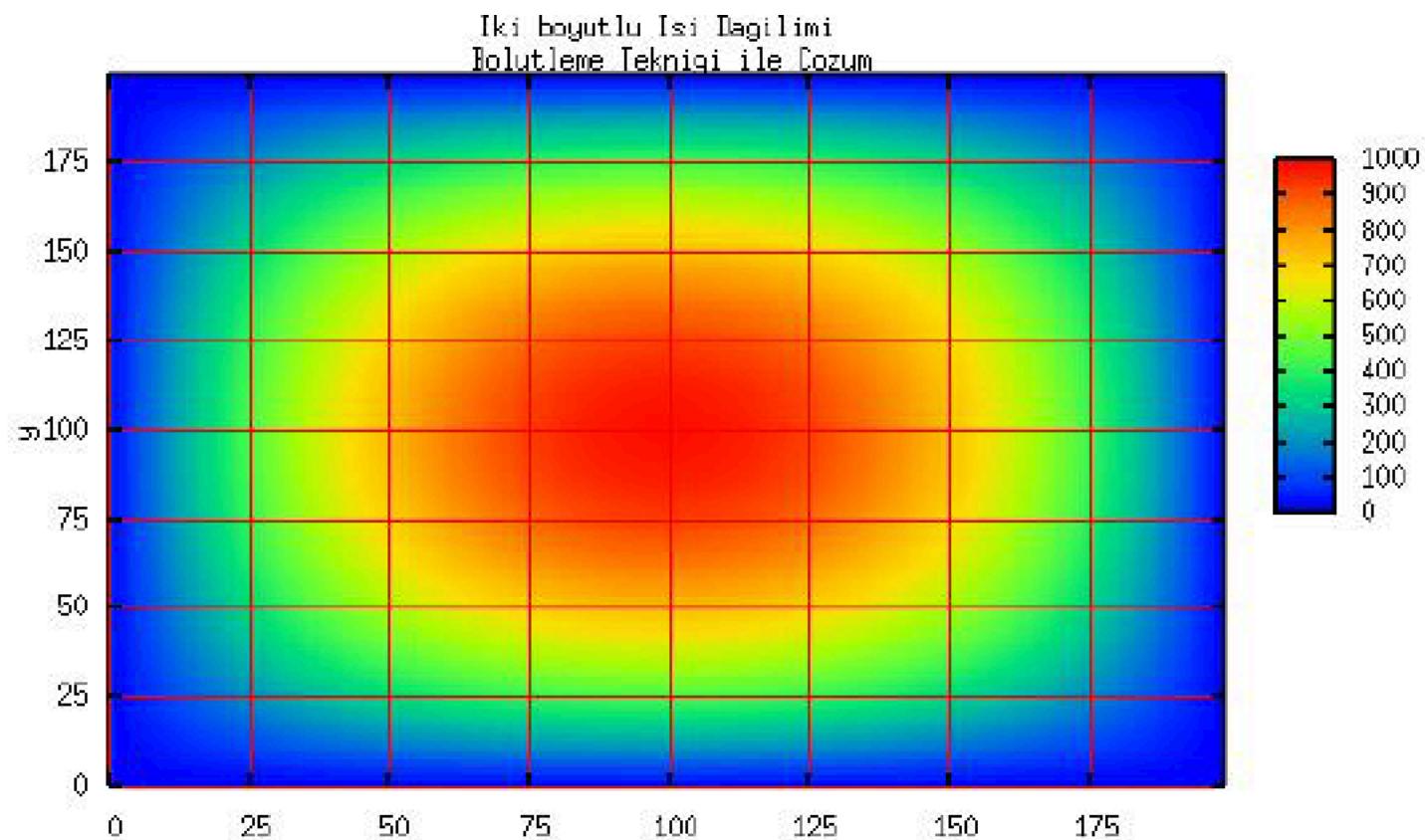
Lo que da un sistema de n ecuaciones con n incógnitas

$$-4u_{i_x,i_y} + u_{i_x-1,i_y} + u_{i_x+1,i_y} + u_{i_x,i_y-1} + u_{i_x,i_y+1} = 0$$

$u_{1,1}$	$u_{1,2}$	$u_{1,3}$	$u_{1,4}$	$u_{1,5}$	$u_{1,6}$	$u_{1,7}$	$u_{1,8}$	$u_{1,9}$
$u_{2,1}$	$u_{2,2}$	$u_{2,3}$	$u_{2,4}$	$u_{2,5}$	$u_{2,6}$	$u_{2,7}$	$u_{2,8}$	$u_{2,9}$
$u_{3,1}$	$u_{3,2}$	$u_{3,3}$	$u_{3,4}$	$u_{3,5}$	$u_{3,6}$	$u_{3,7}$	$u_{3,8}$	$u_{3,9}$
$u_{4,1}$	$u_{4,2}$	$u_{4,3}$	$u_{4,4}$	$u_{4,5}$	$u_{4,6}$	$u_{4,7}$	$u_{4,8}$	$u_{4,9}$
$u_{5,1}$	$u_{5,2}$	$u_{5,3}$	$u_{5,4}$	$u_{5,5}$	$u_{5,6}$	$u_{5,7}$	$u_{5,8}$	$u_{5,9}$
$u_{6,1}$	$u_{6,2}$	$u_{6,3}$	$u_{6,4}$	$u_{6,5}$	$u_{6,6}$	$u_{6,7}$	$u_{6,8}$	$u_{6,9}$

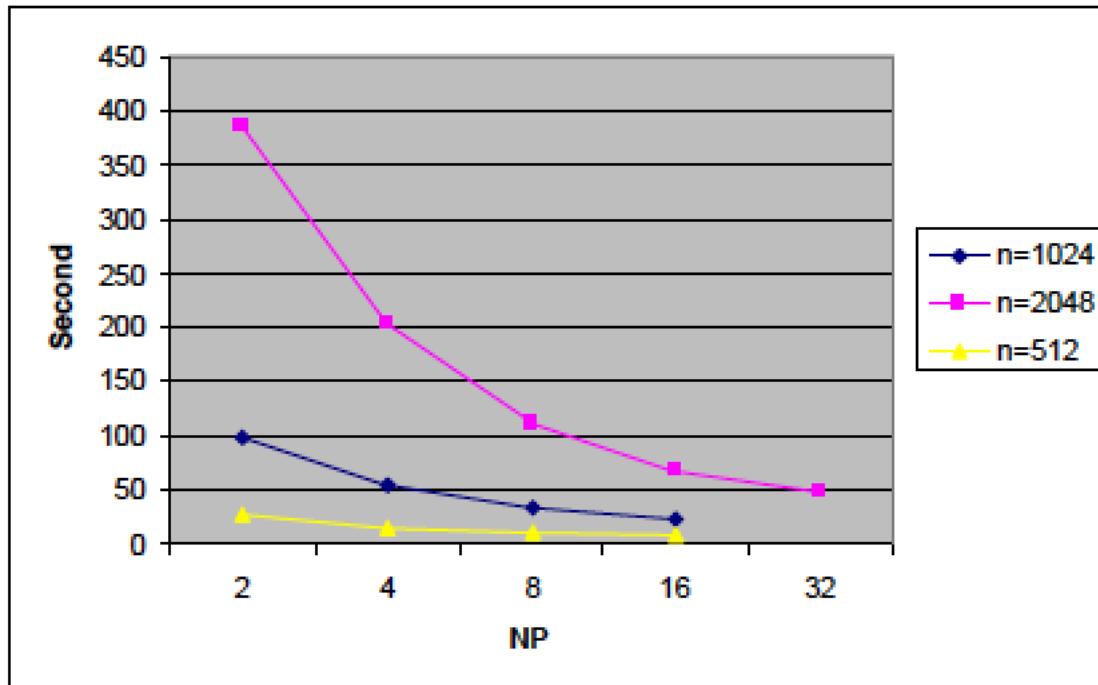
# Ecuación de Laplace (2D):

## Resultados



# Ecuación de Laplace (2D):

Resultados (NP es el número de procesos)



# Solucion de Ecuaciones Diferenciales Parciales: ecuación de Poisson

Soluciones de funciones de potenciales (gravedad, electrostática), describen el estado de equilibrio en procesos de transmision de energía.

La función  $u(x,y)$  es independiente del tiempo (i.e., potencial),  $S(x,y)$  representa a la fuente (masa, carga, o fuente de calor).

Ya que son independientes del tiempo, se llaman de tipo Eliptico.

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = S(x, y)$$

# Solucion de Ecuaciones Diferenciales Parciales

Las soluciones se especifican a las fronteras del dominio (de 1,2,... N dimensiones):

- Condicion de frontera de Dirichlet,  $u(x,y) = f(x,y),$
- Condicion de frontera de Newman

$$\frac{\partial u}{\partial n} = g(x,y),$$

# Solucion de Ecuaciones Diferenciales Parciales: solución numérica

Para la solución de **diferencia finita**, se define una malla de intervalos constantes ( $h_x = h_y$ ) que cubren el dominio.

El numero de puntos de malla es  $N_x \times N_y$

El campo es  $u(x_i, y_j) = u_{i,j}$  y la fuente  $S(x_i, y_j) = S_{i,j}$

Usando la formula de la derivada parcial, la ecuacion diferencial parcial es

$$\frac{1}{h^2} (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4 * u_{i,j}) = S_{i,j}$$

# **Solucion de Ecuaciones Diferenciales Parciales: solución numérica**

Que forman un sistema lineal de ecuaciones, que puede ser resuelto para  $u_{i,j}$

El número de incognitas en este sistema es  $N_x \times N_y$   
en 2D

en 3D para una malla de 1000x1000x1000 puede llegar a ser  $10^9$  ecuaciones

# Solucion de Ecuaciones Diferenciales Parciales: solución numérica iterativa

La solución puede obtenerse iterativamente. Partiendo de un valor inicial  $u_{i,j}$  de los valores en el dominio, el valor del campo en el punto de malla i y j se resuelve partiendo de los valores de los puntos vecinos. El índice k se refiere a la iteración

$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - h^2 S_{i,j})$$

El **método de Jacobi** guarda los valores estimados en cada iteración y los aplica en la siguiente iteración

# Solucion de Ecuaciones Diferenciales Parciales: solución numérica iterativa

$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i+1,j}^k + u_{i-1,j}^{k+1} + u_{i,j+1}^k + u_{i,j-1}^{k+1} - h^2 * S_{i,j})$$

El **método de Gauss-Seidel**, reemplaza el valor del nuevo valor en la misma iteración. Se tienen  $N^2$  ecuaciones para  $N^2$  incógnitas

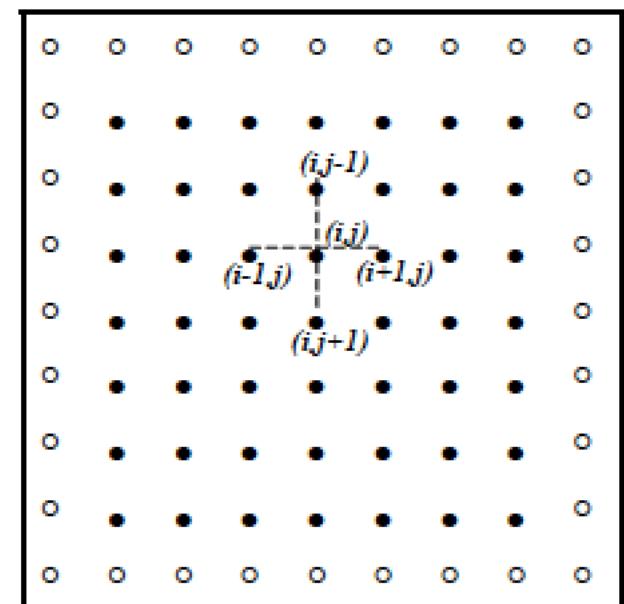
Complejidad:

$T = k m N^2$

$N$  = nodos en cada dimensión

$m$  = operaciones por nodo

$k$  = número de iteraciones



# Solucion de Ecuaciones Diferenciales Parciales: solución numérica iterativa

El **método de Gass-Siedel**, reemplaza el valor del nuevo valor en la misma iteración

$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i+1,j}^k + u_{i-1,j}^{k+1} + u_{i,j+1}^k + u_{i,j-1}^{k+1} - h^2 * S_{i,j})$$

Las iteraciones se repiten hasta convergencia de la diferencia entre los valores antiguos y los nuevos

$$diff = Max ||u_{i,j}^{k+1} - u_{i,j}^k||$$

$$diff \leq \epsilon$$

# Solucion de Ecuaciones Diferenciales Parciales: solución numérica iterativa

Método de Gauss-Seidel (codigo):

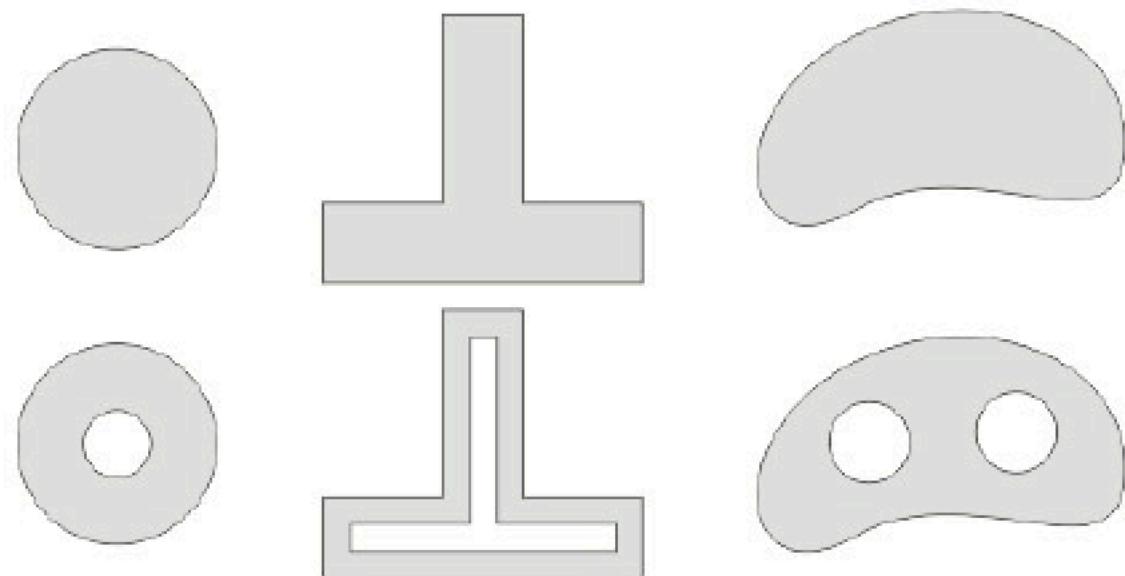
```
do {  
    dmax = 0; // maximum deviation of values u  
    for ( i=1; i<N+1; i++ )  
        for ( j=1; j<N+1; j++ ) {  
            temp = u[i][j];  
            u[i][j] = 0.25*(u[i-1][j]+u[i+1][j]+  
                            u[i][j-1]+u[i][j+1]-h*h*f[i][j]);  
            dm = fabs(temp-u[i][j]);  
            if ( dmax < dm ) dmax = dm;  
        }  
    } while ( dmax > eps );
```

Converge más rápido que jacobi pero necesita más iteraciones

# Descomposición de Dominio

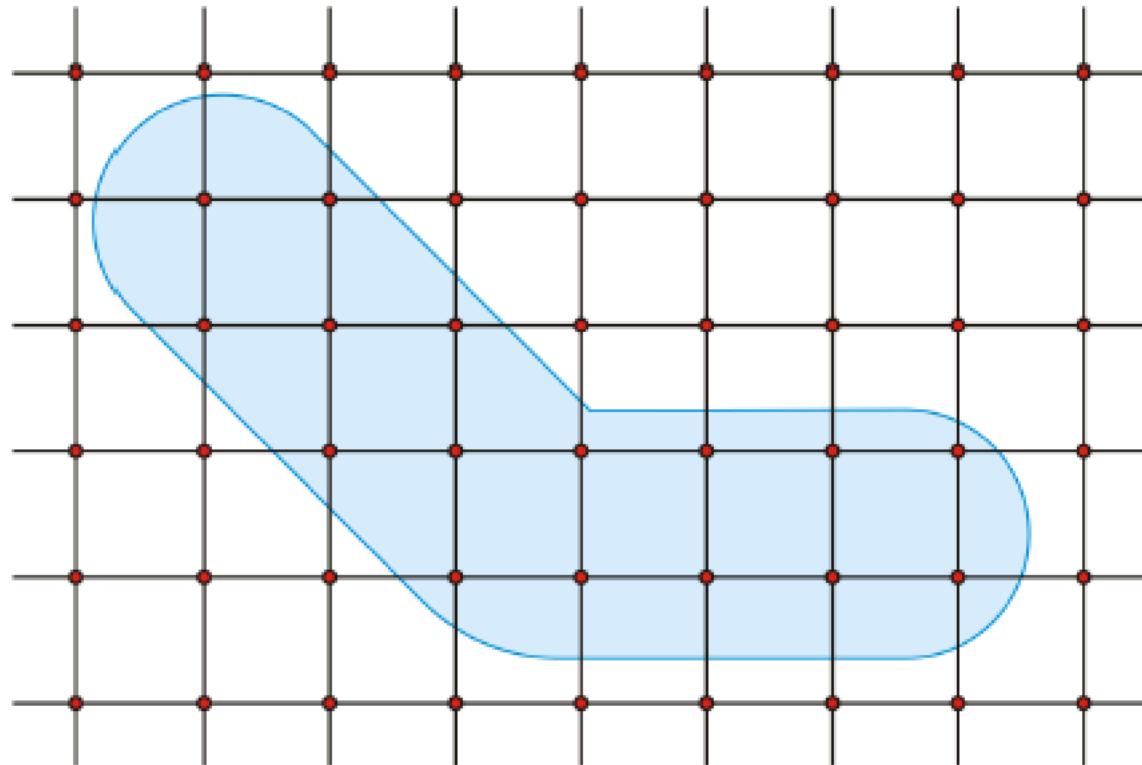
La solución de ecuaciones diferenciales parciales implica una malla numérica para definir el dominio. Normalmente se divide en sub-dominios y se asigna a cada proceso la solución de un sub-dominio

Pero la  
mayoria de  
sistemas no  
son  
rectangulares



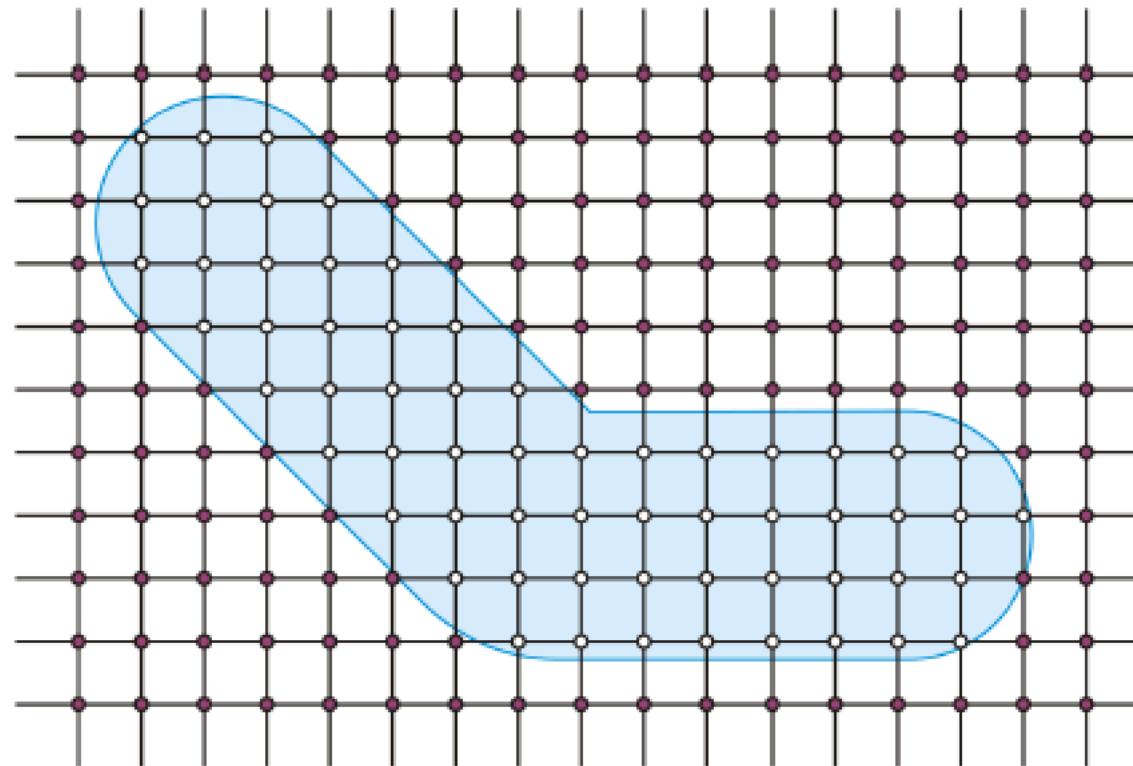
# Descomposición de Dominio

Un dominio irregular puede ser descompuesto, y los puntos fuera del dominio deben ser conocidos



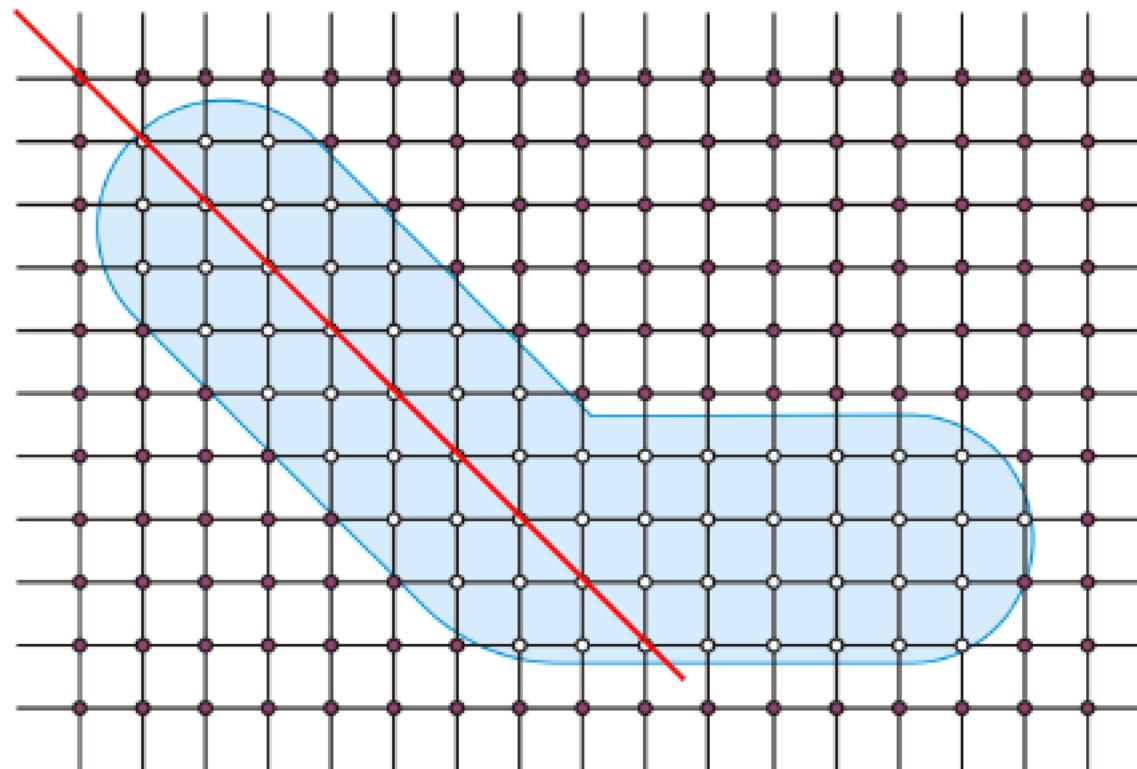
# Descomposición de Dominio

Incrementando el número de puntos es menos eficiente



# Descomposición de Dominio

Existiran algunas simetrias, pero no en todas las areas del dominio



# Descomposición de Dominio

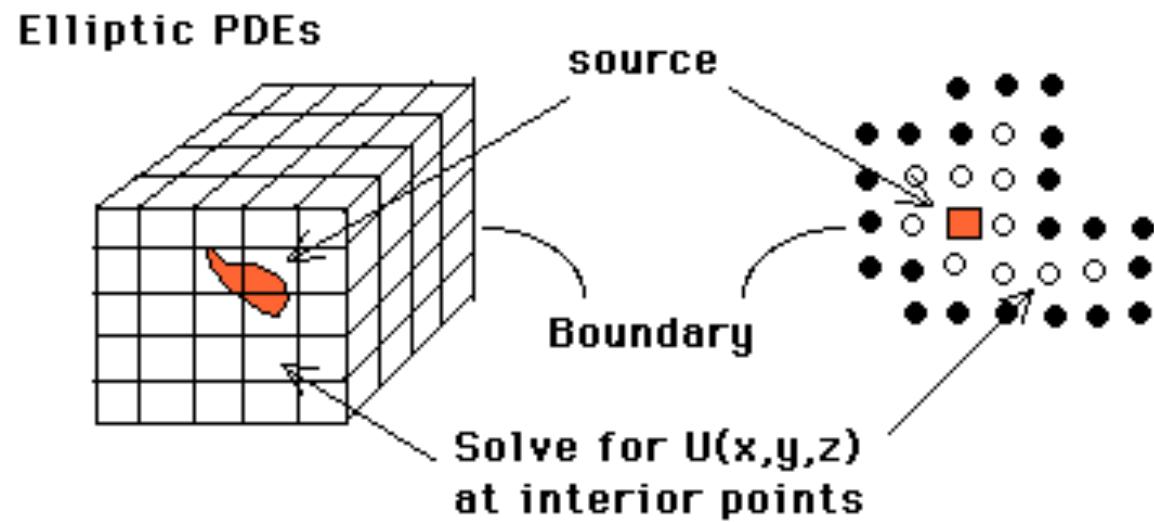
E.g. ecuación de Poisson

$$\frac{\partial^2 u}{\partial^2 x} + \frac{\partial^2 u}{\partial^2 y} + \frac{\partial^2 u}{\partial^2 z} = \rho(x, y, z),$$

Donde  $u(x, y, z)$  es el potencial , y  $\rho(x, y, z)$  es la fuente

# Descomposición de Dominio

Si el dominio es cúbico, se puede descomponer fácilmente, pero es mas difícil si tiene forma arbitraria. Ya que para lograr '**load balancing**', se necesita igual número de puntos de malla en cada proceso



# Descomposición de Dominio

Ecuación parabólica: **ecuación de difusión**

D es un coeficiente de difusión

$$\frac{\partial u}{\partial t} = -\frac{\partial}{\partial x} \left( D \frac{\partial u}{\partial x} \right)$$

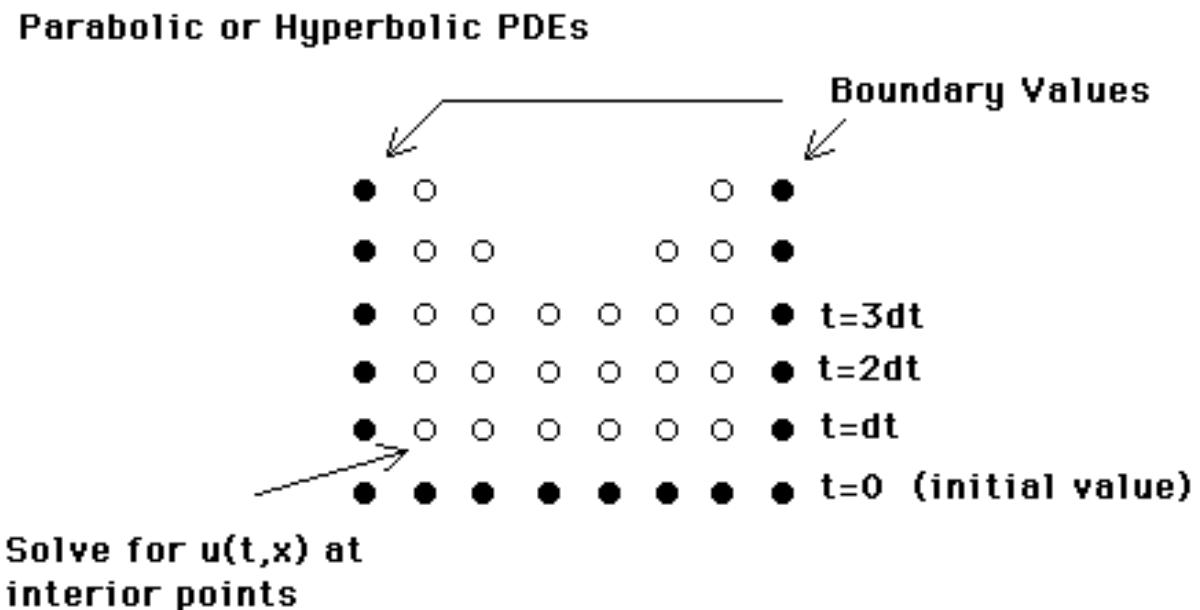
Solución es el valor del campo  $u(x, t)$  a un tiempo t  
dadas las condiciones iniciales en t=0, y sujeto a  
determinadas condiciones de frontera

# Descomposición de Dominio

Ecuación hiperbólica: **ecuación de onda**

$$\frac{\partial^2 u}{\partial t^2} = V^2 \frac{\partial^2 u}{\partial x^2}$$

En este caso, la descomposición del dominio se aplica al espacio, ya que el tiempo debe ser secuencial, y no puede ser paralelizado



# Implementación en paralelo

Se hace asignando un sub-dominio (o máximo un número pequeño de sub-dominios), a cada proceso. Para ello:

- Se debe ejecutar igual trabajo en cada sub-dominio (**load-balancing**)
- Se debe minimizar comunicación **node-to-node** (`MPI_Send`, `MPI_Recv`)

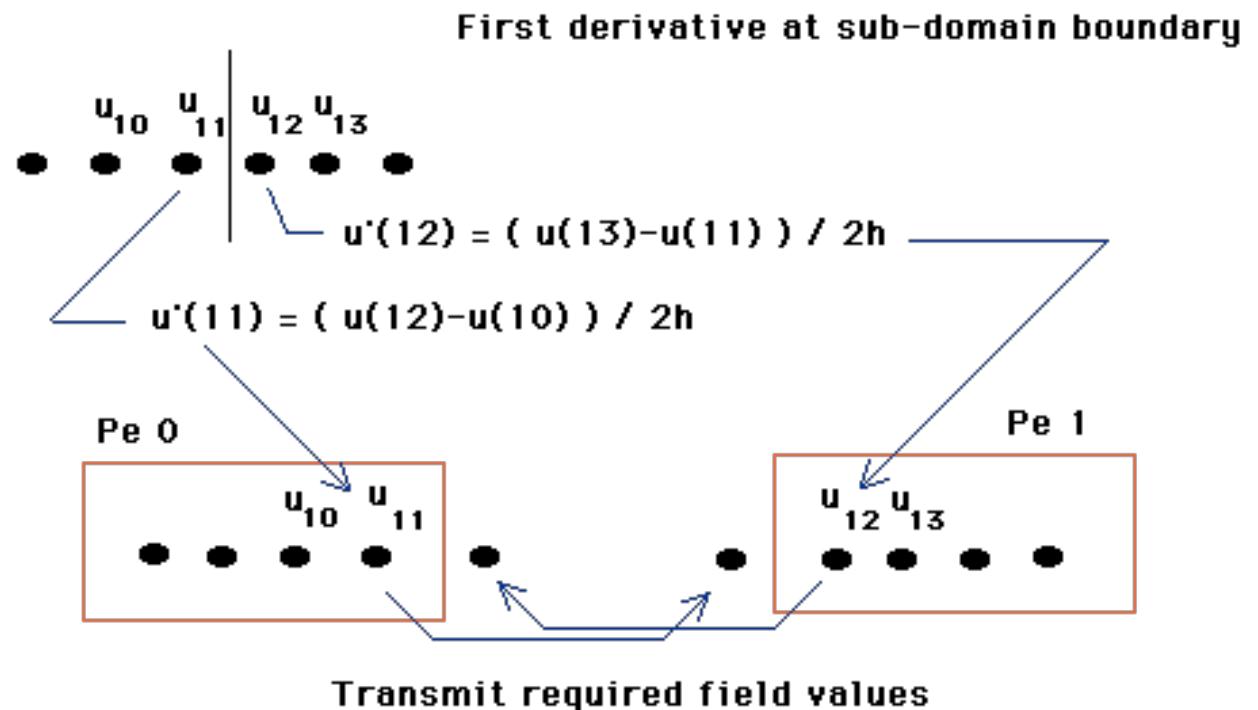
Normalmente, condiciones geométricas o físicas del dominio son preponderantes

**El problema de comunicación *Node-to-node*:**

$$\frac{df}{dx} = \frac{f_{i+1} - f_{i-1}}{2h}$$

$$\frac{d^2 f}{dx^2} = \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2}$$

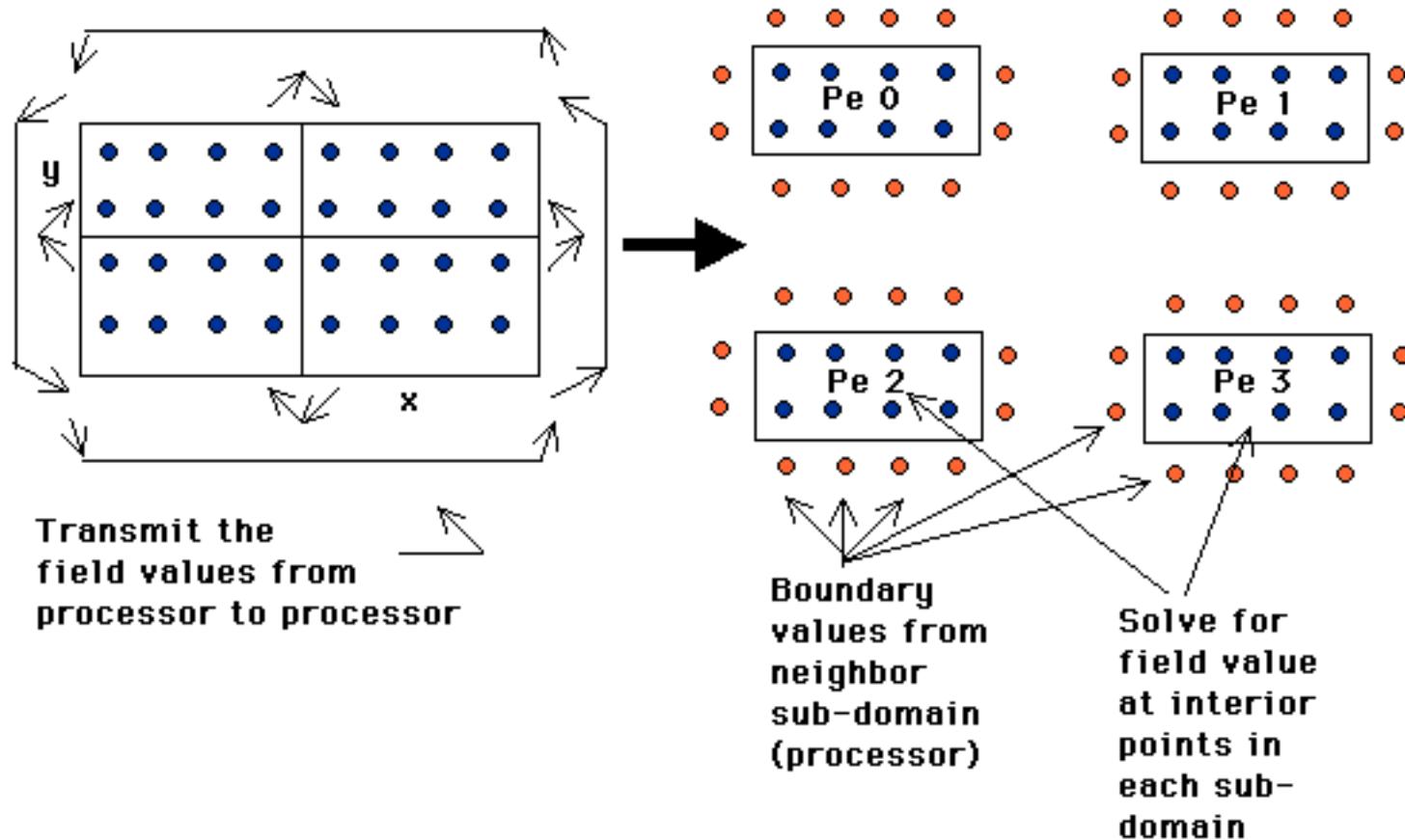
Ej. Para calcular derivadas de las fronteras de los sub-dominios se requieren los valores de los sub-dominios adyacentes, que estan a cargo de otros procesos, lo que requiere comunicación entre ellos.



# El problema de comunicación *Node-to-node*:

En 2 dimensiones:

Domain Decomposition – sub-domains & boundary values



# Ecuación elíptica:

## *Ecuacion de Poisson uni-dimensional*

$$d^2 j(x) / dx^2 = s(x)$$

I.e., la función  $j(x)$  representa un potencial eléctrico de una distribución de carga  $s(x)$ . El dominio es uni-dimensional ( $x=[0,1]$ ). Las condiciones de frontera de tipo Dirichlet  $j(0)$  y  $j(1)$ . La fuente es un Gaussiano centrado en el punto medio

# Ecuación elíptica: *Ecuacion de Poisson uni-dimensional*

El dominio de la ecuación de Poisson puede ser discretizado con N número de puntos de malla.

$$x_i = (i-1) h, \quad h = 1.0 / (N - 1),$$

Lo que representa un sistema de ecuaciones a ser resuelto por un método iterativo, con

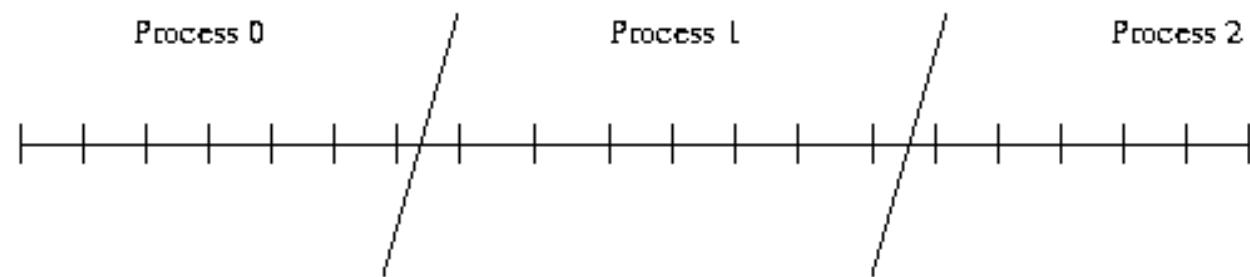
$$(j_{i+1} - 2j_i + j_{i-1}) / h^2 = S_i$$

$$j_i = 1 / 2 (j_{i+1} + j_{i-1} - h^2 S_i)$$

Hasta que los cambios en  $j_i$  sean menores que una tolerancia

# Ecuación elíptica: *Ecuacion de Poisson uni-dimensional*

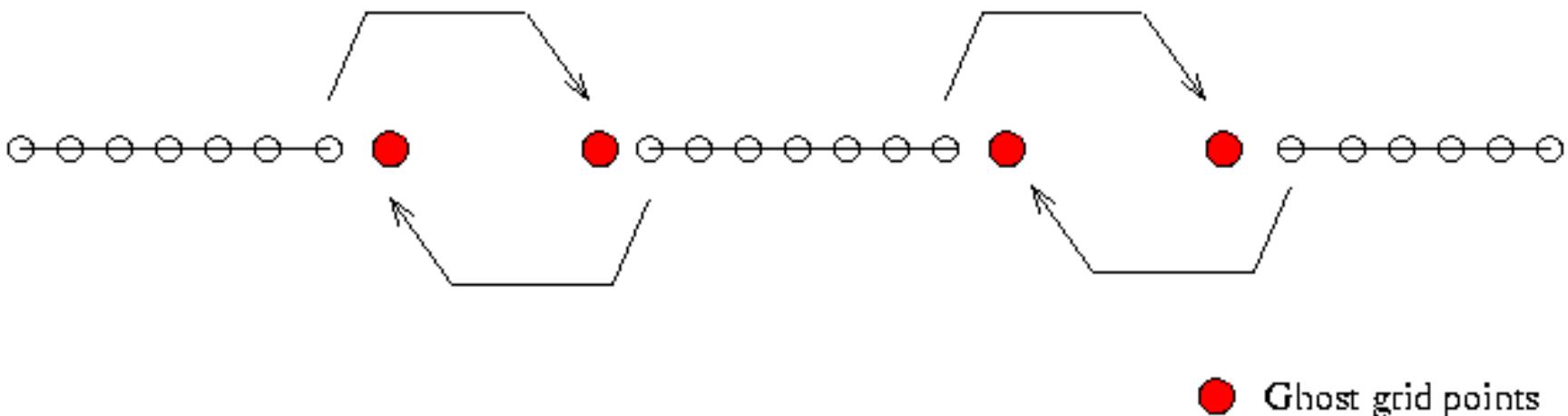
La descomposición del dominio consiste en dividir la malla original entre los procesos con un número equivalente de puntos de malla para un buen load-balance



# Ecuación elíptica: *Ecuacion de Poisson uni-dimensional*

Lo que resulta en el siguiente ordenamiento de cada proceso

1-D sub-domain definition



# Ecuación elíptica: *Ecuacion de Poisson bi-dimensional*

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = S(x, y)$$

En forma diferencial finita

$$\frac{1}{h^2} (u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4 * u_{i,j}) = S_{i,j}$$

# Ecuación elíptica: *Ecuacion de Poisson bi-dimensional*

Sistema de ecuaciones que se resuelve de forma iterativa

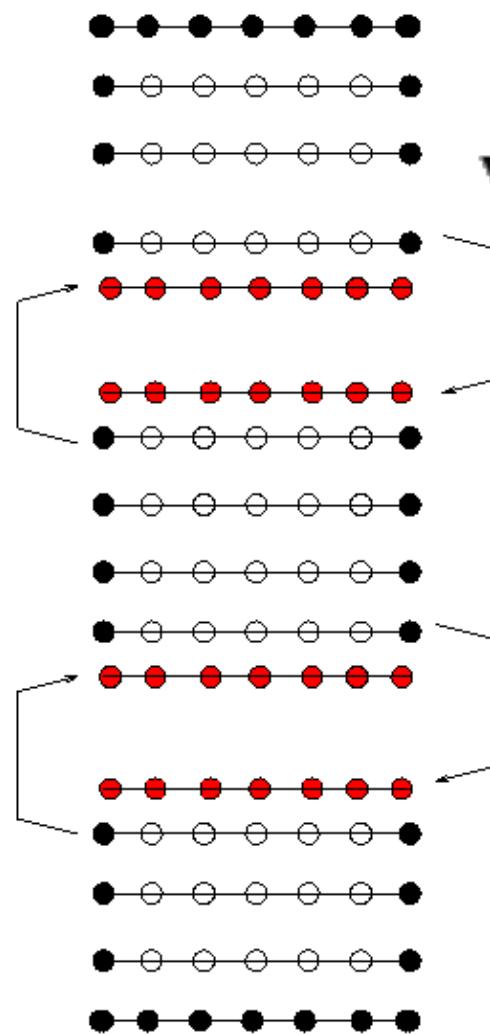
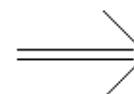
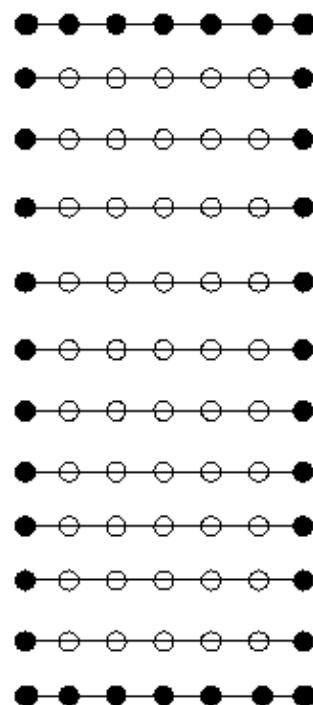
$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i+1,j}^k + u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k - h^2 S_{i,j})$$

Hasta que se obtenga la convergencia de

$$diff = Max||u_{i,j}^{k+1} - u_{i,j}^k||$$

# Ecuación elíptica: *Ecuacion de Poisson bi-dimensional*

2-D Domain Decomposition



$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = S(x, y)$$

# Ecuacion parabólica de difusión:

$$\partial u(x,y,t) / \partial t = \partial^2 u(x,y,t) / \partial x^2 + \partial^2 u(x,y,t) / \partial y^2 + s(x,y,t)$$

En forma diferencial finita

$$(u^{k+1}_{ij} - u^k_{ij}) / dt = \\ 1/2 (\delta_x^2 u^{k+1}_{ij} + \delta_y^2 u^{k+1}_{ij} + \delta_x^2 u^k_{ij} + \delta_y^2 u^k_{ij}) + 1/2 (S^{k+1}_{ij} + S^k_{ij})$$

Donde:

$$\delta_x^2 u^{k+1}_{ij} = (u^{k+1}_{i+1,j} - 2u^{k+1}_{ij} + u^{k+1}_{i-1,j}) / h^2$$

$$\delta_y^2 u^{k+1}_{ij} = (u^{k+1}_{i,j+1} - 2u^{k+1}_{ij} + u^{k+1}_{i,j-1}) / h^2$$

# Ecuacion parabólica de difusión:

Con  $r = dt/(2h^2)$

$$u^{k+1}_{ij} = ( r(u^{k+1}_{i+1,j} + u^{k+1}_{i-1,j} + u^{k+1}_{i,j+1} + u^{k+1}_{i,j-1}) + d^k_{ij} ) / (1+4r)$$

Se busca la solución  $u^{k+1}_{ij}$ , donde:

$$d^k_{ij} = r(u^k_{i+1,j} + u^k_{i-1,j} + u^k_{i,j+1} + u^k_{i,j-1}) + dt(S^{k+1}_{ij} + S^k_{ij}) / 2$$

# Ecuacion hiperbólica de onda:

Una cadena oscilante se describe con una ecuación de onda

$$\delta^2\Phi/\delta t^2 = -c^2 \delta^2\Phi/\delta x^2$$

que representa un sistema que evoluciona en el tiempo



# Ecuacion hiperbólica de onda:

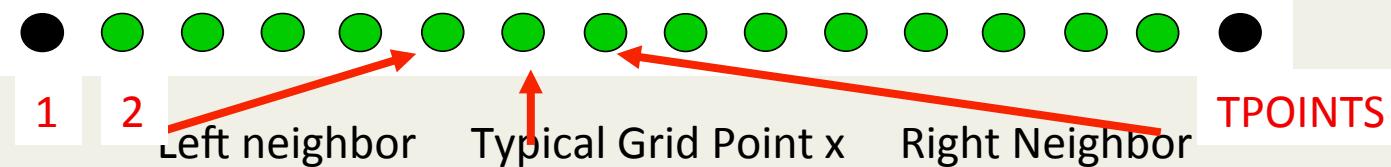
Discretizando:

$$(u(x,t+1) - 2u(x,t) + u(x,t-1)) / \delta t^2 = -c^2 (u(x+1,t) - 2u(x,t) + u(x-1,t)) / \delta x^2$$

O tambien, para calcular  $u$  en un paso en el futuro, basado en  $x$  actual y un paso en el pasado

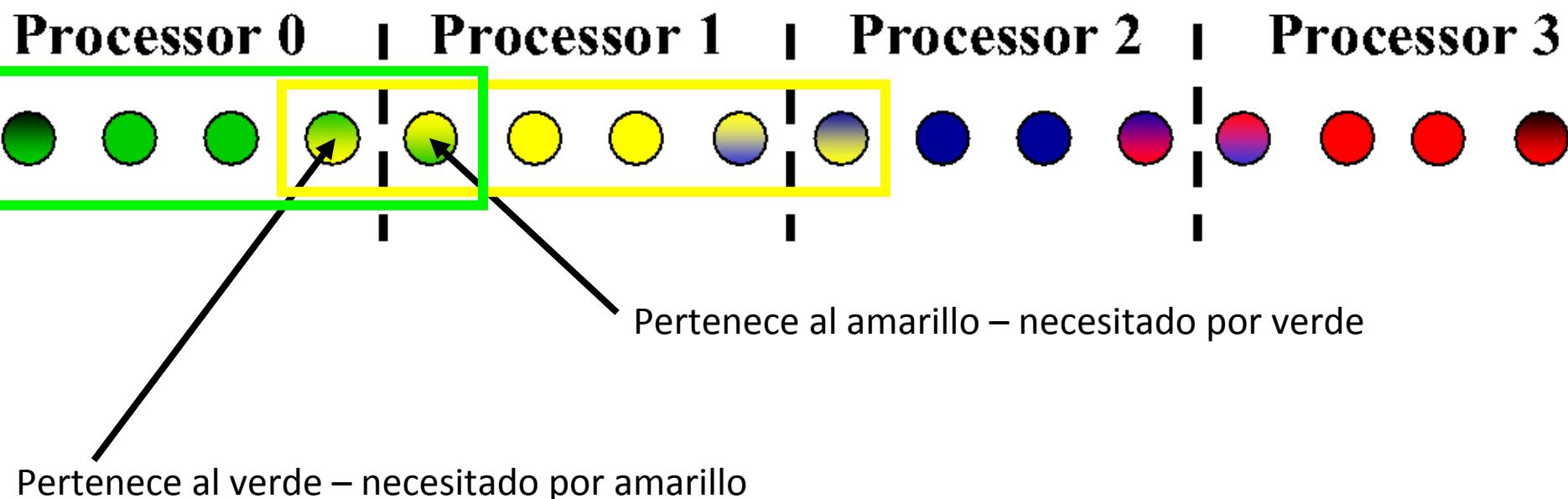
$$u(x,t+1) = 2u(x,t) - u(x,t-1) - (u(x+1,t) - 2u(x,t) + u(x-1,t)) (c^2 \delta t^2 / \delta x^2)$$

Sequential One Dimensional Wave Equation



# Ecuacion hiperbólica de onda:

Esquema de algoritmo en paralelo



Necesita comunicación P2P para  
intercambiar puntos de frontera con vecinos

# Programa en serie para resolver la ecuacion de onda:

```
#define MAXPOINTS 1000
#define MAXSTEPS 1000
#define MINPOINTS 20
#define PI 3.14159265

int main(int argc, char *argv[])
{
    init_param();
    init_line();
    update();
    printfinal();
}
```

## Inicialización de puntos en línea, de acuerdo a función sin()

```
void init_line(void) {
    int i, j;
    double x, fac, k, tmp;
    /* Calculate initial values based
       on sine curve */
    fac = 2.0 * PI;
    k = 0.0;
    tmp = tpoints - 1;
    for (j = 1; j <= tpoints; j++) {
        x = k/tmp;
        values[j] = sin (fac * x);
        k = k + 1.0;
    }
    /* Initialize old values array */
    for (i = 1; i <= tpoints; i++)
        oldval[i] = values[i];
}
```

## Actualizar los valores en una linea un número determinado de veces

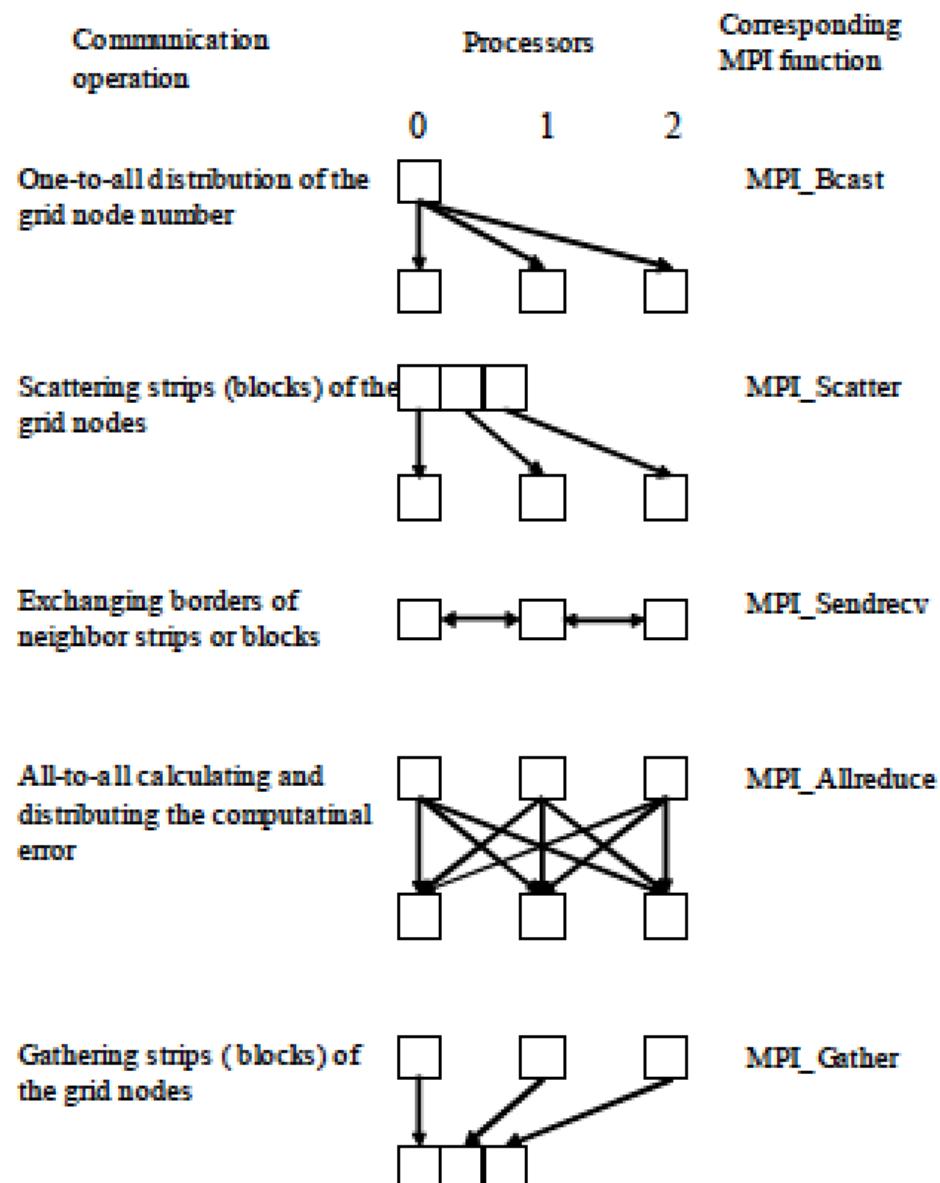
```
void update()
{
    int i, j;
    /* Update values for each time step */
    for (i = 1; i <= nsteps; i++) {
        /* Update points along line for this time step */
        for (j = 1; j <= tpoints; j++) {
            /* global endpoints */
            if ((j == 1) || (j == tpoints))
                newval[j] = 0.0;
            else
                do_math(j);
        }
        /* Update old values with new values */
        for (j = 1; j <= tpoints; j++) {
            oldval[j] = values[j];
            values[j] = newval[j];
        }
    }
}
```

# Calculo de nuevos valores a partir de la ecuacion de onda

```
void do_math(int i)
{
    double dtime, c, dx, tau, sqtau;

    dtime = 0.3;
    c = 1.0;
    dx = 1.0;
    tau = (c * dtime / dx);
    sqtau = tau * tau;
    newval[i] = (2.0 * values[i]) -oldval[i]
    + (sqtau * (values[i-1] - (2.0 *
    values[i]) + values[i+1])));
}
```

# Comunicación de data para la solución de ecuaciones diferenciales



# Algoritmos Paralelos

```
void update(int left, int right) {
.....
/* Update values for each point along string */
for (i = 1; i <= nsteps; i++) {
    /* Exchange data with "left-hand" neighbor */
    if (first != 1) {
        MPI_Send(&values[1], 1, MPI_DOUBLE, left, RtoL, MPI_COMM_WORLD);
        MPI_Recv(&values[0], 1, MPI_DOUBLE, left, LtoR, MPI_COMM_WORLD,
                 &status);
    }
    /* Exchange data with "right-hand" neighbor */
    if (first + npoints -1 != TPOINTS) {
        MPI_Send(&values[npoints], 1, MPI_DOUBLE, right, LtoR, MPI_COMM_WORLD);
        MPI_Recv(&values[npoints+1], 1, MPI_DOUBLE, right, RtoL,
                 MPI_COMM_WORLD, &status);
    }
    /* Update points along line */
    .....
}
}
```