Restauración de imágenes: un caso sencillo

En este ejemplo se pretende aplicar una técnica para **restaurar una imagen a partir de la salida que se obtiene después de aplicar** un filtro de detección de aristas como el **filtro de Laplace**.

Imagen original

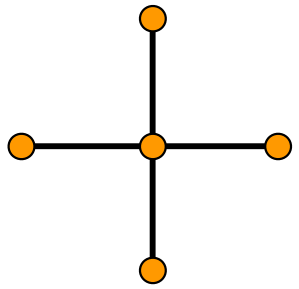
Detección de aristas

Imagen restaurada

Restauración de imágenes: un caso sencillo (detección de aristas)

- Una **arista local** es un **píxel** cuyo nivel de gris **difiere significativamente** del nivel de gris de algunos **píxeles de su entorno**. Es decir, hay diferencia de contraste local.
- Las aristas locales se detectan midiendo la **tasa de cambio de los tonos de gris de su entorno**. Vamos a utilizar, para ello, el operador **Laplaciano**.
- El Filtro de Laplace consiste en aplicar la siguiente máscara sobre los píxeles de la imagen:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$



Restauración de imágenes: un caso sencillo (detección de aristas)

- Si los píxeles de la imagen original están almacenados en el array:

```
int img_data[row][col];
```

- Entonces, la aplicación del filtro de Laplace equivale a:

```
int img_mod[row][col];
for (i=1; i<row-1 ;i++) {
    for (j=1; j<col-1; j++){
       img_mod[i][j] = img_data[i-1][ j ] + img_data[i+1][ j ] +
                       img_data[ i ][j-1] + img_data[ i ][j+1] -
                     4*img_data[ i ][ j ];
```

- La aplicación de esta máscara en los **extremos de la imagen** se puede realizar de varias formas:
  - Teniendo en cuenta ceros en las posiciones necesarias,
  - Replicando las filas y columnas correspondientes de píxeles (primera y última).

Restauración de imágenes: un caso sencillo (reconstrucción de la imagen)

- Es posible **reconstruir la imagen original** a partir de las aristas localizadas por la máscara de Laplace realizando de forma iterativa las operaciones:

```
double img_old[row+2][col+2], img_new[row+2][col+2];
int img_edge[row+2][col+2];
for (i=1; i<row+1 ;i++)
    for (j=1; j<col+1; j++)
       img_new[i][j] = 0.25*(img_old[i-1][ j ] + img_old[i+1][ j ] +
                             img_old[ i ][j-1] + img_old[ i ][j+1] -
                            img_edge[ i ][ j ]);
```

- Donde
  - **img_old** e **img_new** son los valores de la imagen al principio y al final de cada iteración.
  - **img_edge[1:row][1:col]** contiene los valores de la imagen una vez aplicada la máscara de Laplace.
  - En la primera iteración se inicializa **img_old = img_edge**.

Restauración de imágenes: un caso sencillo (Algoritmo de reconstrucción)

- Copiar los píxeles obtenidos tras la aplicación de la máscara de Laplace en **img_edge[1:row][1:col].** Las fronteras se dejan a 0:
- Inicializar **img_old** como **img_edge**:

```
for (i=0; i<row+2 ;i++)
    for (j=0; j<col+2; j++)
       if (i==0 || i==row+1 || j==0 || j==col+1) img_edge[i][j] = 0;
       else img_edge[i][j] = img_mod[i-1][j-1]);
       img_old[i][j] = img_edge[i][j];
```

- Iterar sobre **img_new** e **img_old**:

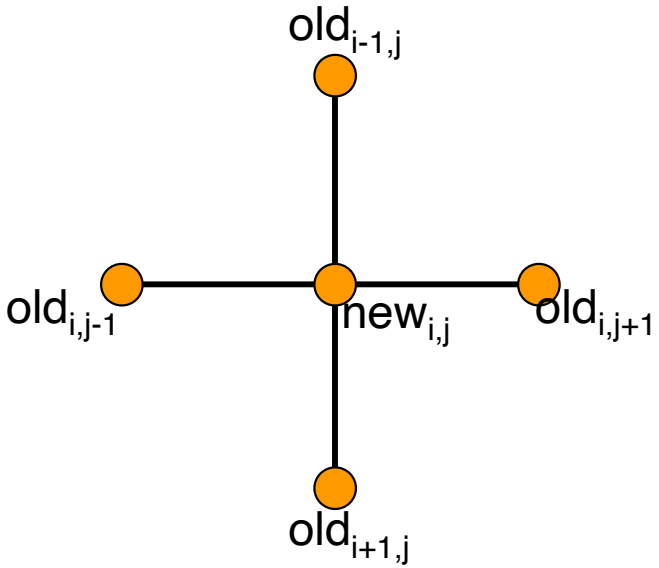
```
for (k=0; k<IterMax; k++)
   for (i=1; i<row+1 ;i++)
       for (j=1; j<col+1; j++)
          img_new[i][j] = 0,25*(img_old[i-1][ j ] + img_old[i+1][ j ] +
                                 img_old[ i ][j-1] + img_old[ i ][j+1] -
                                img_edge[ i ][ j ]);
   for (i=1; i<row+1 ;i++)
       for (j=1; j<col+1; j++)
          img_old[i][j] = img_new[i][ j ];
```

Restauración de imágenes: un caso sencillo (Paralelización del algoritmo)

- La primera consideración a realizar antes de estudiar el paralelismo de este algoritmo radica en el esquema de dependencias en el bucle principal del algoritmo:

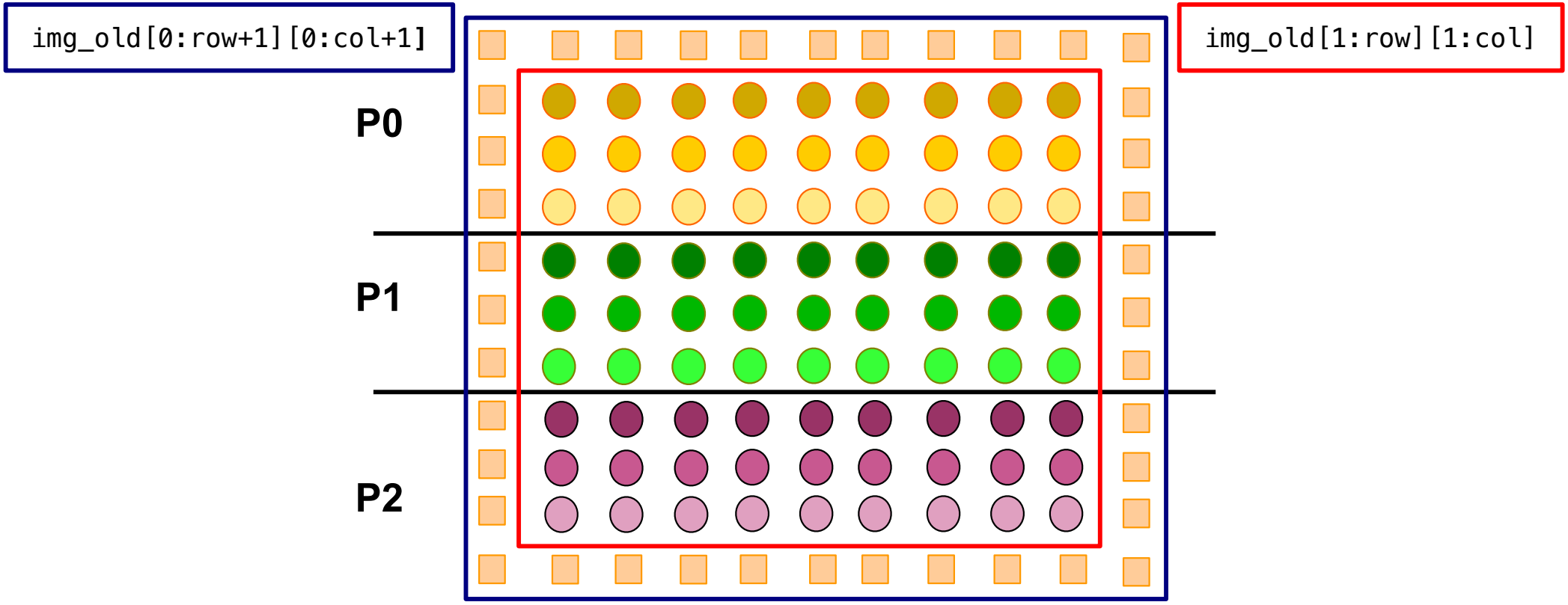
```
for (k=0; k<IterMax; k++)
   for (i=1; i<row+1 ;i++)
       for (j=1; j<col+1; j++)
          img_new[i][j] = 0,25*(img_old[i-1][ j ] + img_old[i+1][ j ] +
                                img_old[ i ][j-1] + img_old[ i ][j+1] -
                                img_edge[ i ][ j ]);
```

- Cada pixel se actualiza a partir de los píxeles de sus 4 vecinos:



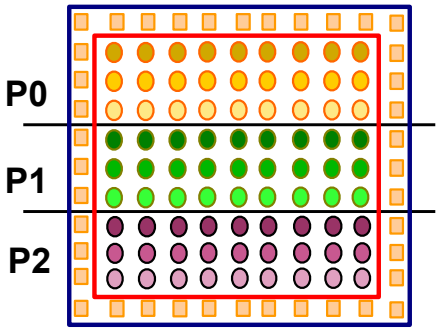
Restauración de imágenes: un caso sencillo (Paralelización del algoritmo)

- Para nuestra implementación paralela consideraremos una descomposición de los píxeles útil para una topología de anillo bidireccional de manera que cada proceso se encargará del cálculo de una **porción de filas consecutivas de la imagen**, tal y como indica la figura siguiente. Por simplicidad tomaremos como ejemplo una imagen de tamaño 9x9 y 3 procesos.

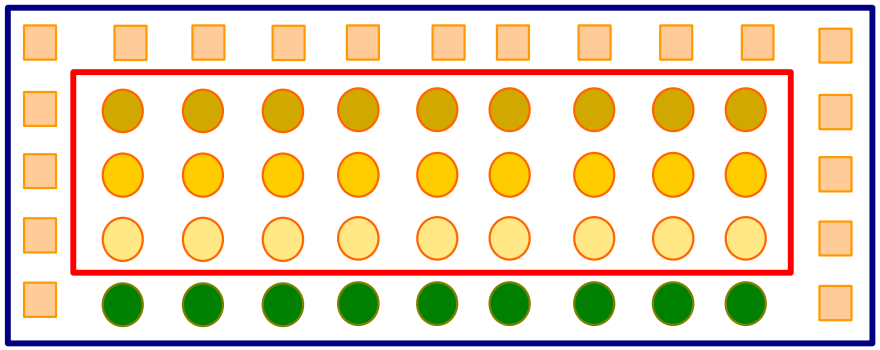


Restauración de imágenes: un caso sencillo (Paralelización del algoritmo)

- Teniendo en cuenta el esquema de dependencias, cada proceso necesitará el siguiente almacenamiento:



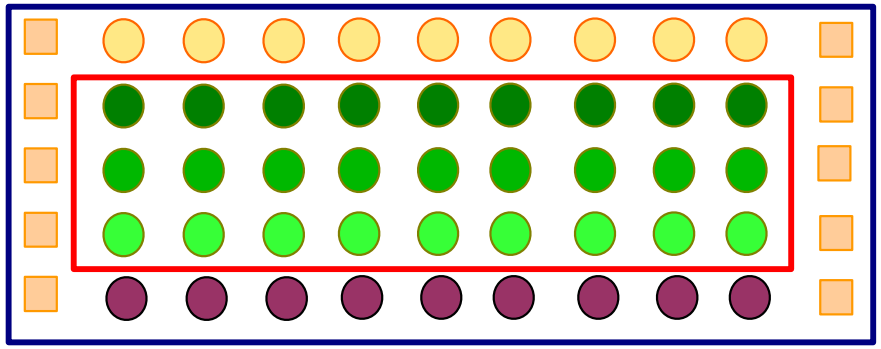
P0



`img_old[1:rowlocal][1:col]`

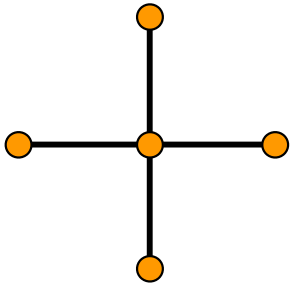
`img_old[0:rowlocal+1][0:col+1]`

P1

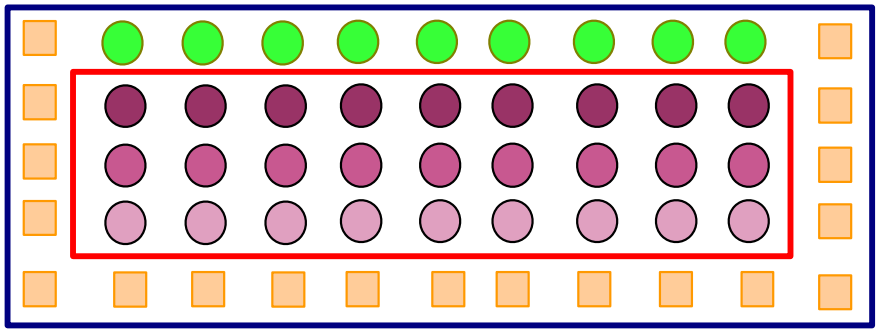


`img_old[1:rowlocal][1:col]`

img_old[0:rowlocal+1][0:col+1]



P2

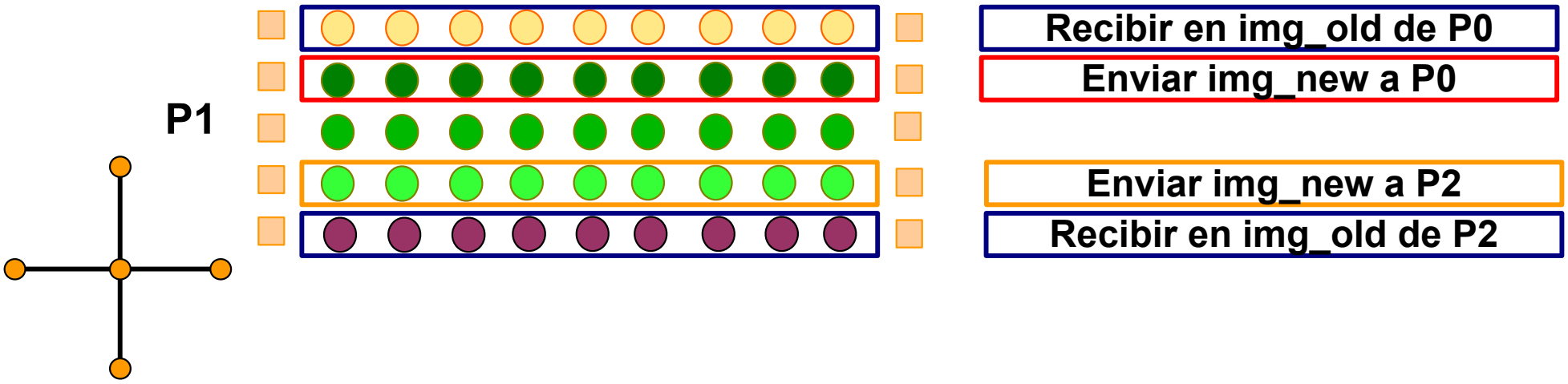


`img_old[1:rowlocal][1:col]`

`img_old[0:rowlocal+1][0:col+1]`

Restauración de imágenes: un caso sencillo (Paralelización del algoritmo)

- Al finalizar cada iteración, cada proceso debe comunicar sus valores frontera:
  - Enviar su primera fila de píxeles al proceso anterior: **img_new[1][1:col].**
  - Enviar su última fila de píxeles al proceso posterior: **img_new[rowlocal][1:col].**
  - Recibir del proceso posterior **col** píxeles y colocarlos en **img_old[rowlocal+1][1:col].**
  - Recibir del proceso anterior **col** píxeles y colocarlos en **img_old[0][1:col].**
- El primer proceso y el último envían y reciben un solo mensaje.



Restauración de imágenes: un caso sencillo (Paralelización del algoritmo)

- Si este intercambio de píxeles no se realiza en cada iteración, el resultado es una restauración con ciertos errores en las fronteras correspondientes a cada proceso.
- Por ejemplo, si aplicamos este algoritmo a la imagen que se muestra, sin intercambio de píxeles entre 4 procesos el resultado es: