Práctica 2

Sesión 2

Objetivos

- Conocer cómo se genera un fractal a través del método de Newton
- Entender los algoritmos maestro-esclavo
  - Maestro no colabora
  - Maestro colabora
- Modificar una implementación paralela en la que el maestro no colabora, a otra en la que el maestro colabora
- Conocer las comunicaciones punto a punto no bloqueantes:
  - **MPI_Irecv** (recepción)
  - **MPI_Wait** (espera)
  - **MPI_Test** (test)

Fractales (I)

- ¿Qué es un fractal? ¿Qué propiedades tiene?
  - Objeto geométrico cuya estructura se repite a diferentes escalas.
  - En algunos casos, su cálculo puede conllevar un coste computacional considerable.
  - En esta práctica se va a partir de un programa paralelo en MPI (newton.c) para calcular fractales de Newton, el cual habrá que modificar convenientemente.

Fractales (II)

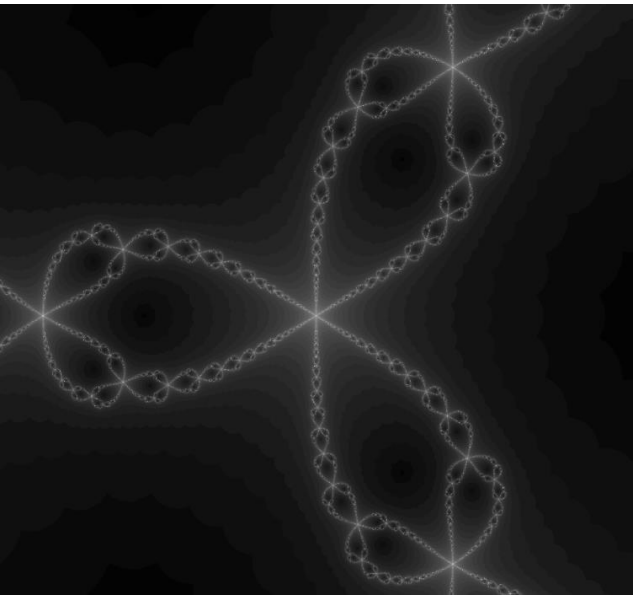


Fig 3. $f(z)=z^3-1$.

```
Algoritmo
Para y = y1,..., y2
   Para x = x1,..., x2
      col = num_iteraciones_Newton(funcion, x, y, tol, maxiter)
      Pintar el punto (x,y) con el color col
   fin_para
fin_para
```

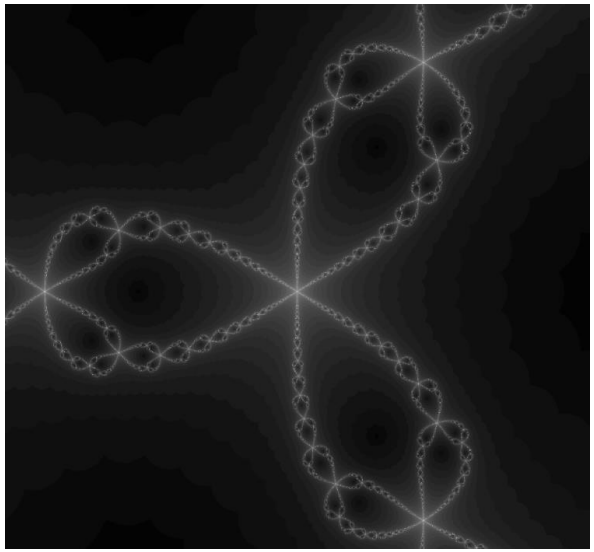
- El método de Newton es un método iterativo que permite obtener una raíz compleja de la ecuación f(z)=0, partiendo de un punto $z=(x_0,y_0)$:
  - Mediante una fórmula, se genera una sucesión puntos que puede converger a la solución.
  - El método para cuando dos puntos consecutivos distan menos que tol.
  - Mediante maxiter se limita el número máximo de iteraciones.
- Partiendo del punto (x_0,y_0),la función **num_iteraciones_Newton** (x_0,y_0), devuelve:
  - Si ha habido convergencia, devuelve el número de iteraciones alcanzado.
  - Si no la ha habido, devuelve el valor **maxiter**.
- El máximo número de iteraciones corresponde al color blanco.

Ejercicio 1

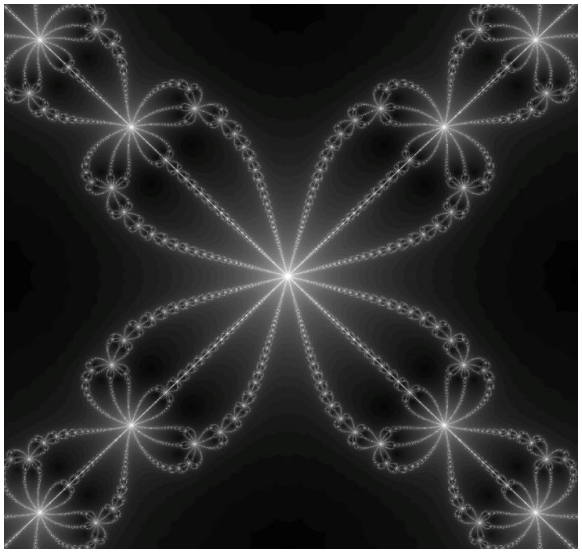
- Compila el programa newton.c, usando mpicc y el parámetro **-lm**
- En la ejecución usa el script mpiexec seguido del ejecutable y las opciones que a continuación se detallan
- Para generar los diferentes fractales utiliza las opciones:
  - -c1, -c2, -c3, -c4: genera 4 fractales para las pruebas básicas
  - -c5: genera un fractal para para analizar las prestaciones de las implementaciones paralelas, por ser más costoso que el resto
- El nombre de la imagen por defecto es newton.pgm. Este nombre se puede cambiar con la opción –o.
- Por ejemplo, para que se genere el fractal –c1 y que el fichero de salida generado sea refc1.pgm, ejecutaríamos:

  mpiexec newton –orefc1 –c1
- Recuerda que, para comprobar el correcto funcionamiento de un programa paralelo, las imágenes generadas por el código original y el modificado deben coincidir, usando para ello, por ejemplo, el comando **cmp**

Fractales opciones –c1 a –c5



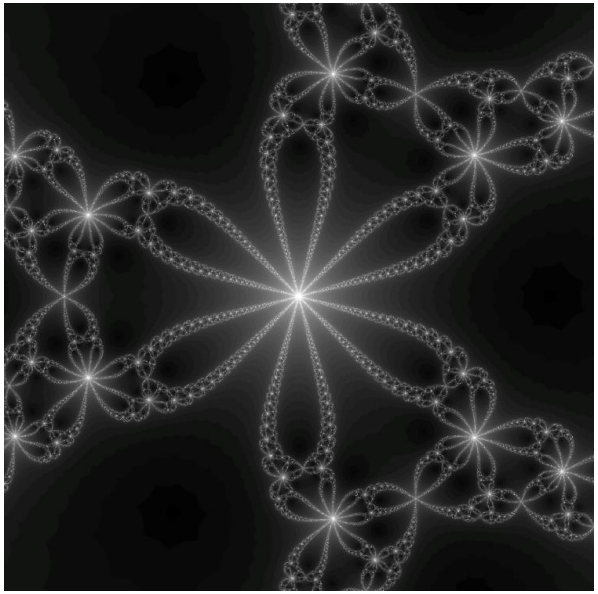
-c1



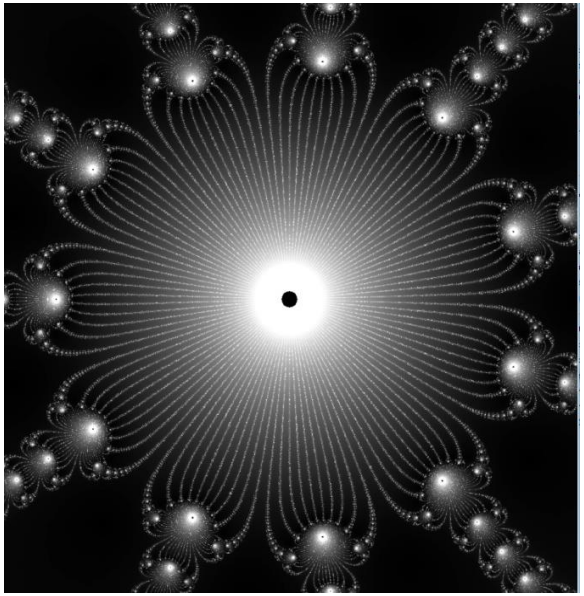
-c2

Nota:

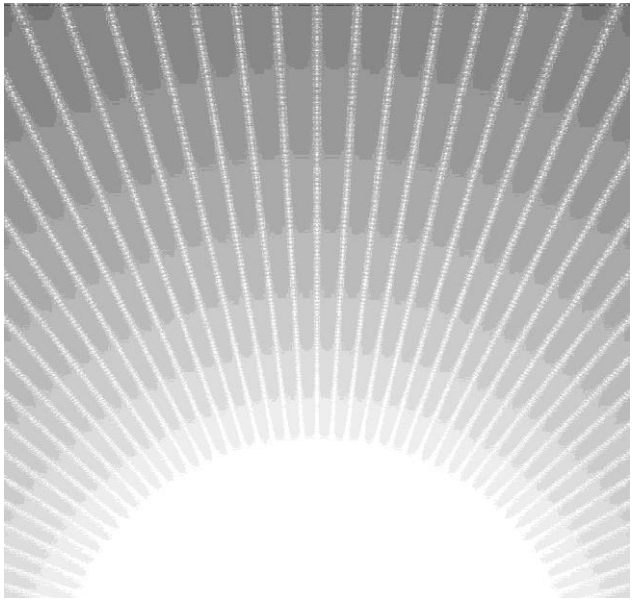
- Los puntos cuyo cálculo tiene un mayor coste computacional son los puntos en blanco
- -c5 es el fractal más costoso



-c3



-c4



-c5

Fig.4 Algoritmo maestro-trabajador clásico (maestro no colabora)

Si yo=0 **entonces** (soy el maestro)
    siguiente_fila <- 0
    Para proc = 1 ... np
        P0 envía al proceso **proc** una petición para que calcule la fila del fractal **siguiente_fila**
        siguiente_fila <- siguiente_fila + 1
    fin_para
    filas_hechas <- 0
    Mientras filas_hechas < filas_totales
        recibe de cualquier proceso una fila calculada
        proc <- proceso que ha enviado el mensaje
        num_fila <- número de fila
        envíar al proceso **proc** petición de hacer la fila número **siguiente_fila**
        siguiente_fila <- siguiente_fila + 1
        copiar fila calculada a su sitio, que es la fila **num_fila** de la imagen
        filas_hechas <- filas_hechas + 1
    fin_mientras
si_no (soy un trabajador)
    Recibir en **num_fila** la fila que tiene que procesar
    Mientras num_fila < filas_totales
        Procesar la fila número **num_fila**
        Envíar la fila recién calculada al maestro
        Recibir en **num_fila** el número de fila que tiene que procesar
    fin_mientras
fin_si

- **siguiente_fila (maestro)**:
  Nº de fila que va a ser procesada
- **filas_hechas (maestro)**:
  Nº de filas que se han procesado
- **num_fila**:
  - **Esclavo**:
    Nº de fila enviada
  - **Maestro**:
    Nº de fila recibida

Ejercicio 2: entender código maestro no colabora

```
149  if (me == 0) { /* CODE FOR THE MASTER */
150    /* Initial distribution of work */
151    next_row = 0;
152    for ( proc = 1 ; proc < np ; proc++ ) {
153      MPI_Send(&next_row, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
154      next_row++;
155    }
156    /* While there are rows to be received */
157    for ( rows_done = 0 ; rows_done < h ; rows_done++ ) {
158      /* Receive a computed row from any process */
159      MPI_Recv(B, w, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
160               &status);
161      /* Get the process index and the row number */
162      proc = status.MPI_SOURCE;
163      /* The row number is in the message TAG */
164      num_row = status.MPI_TAG;
165      /* Ask that process to compute another row */
166      MPI_Send(&next_row, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
167      next_row++;
168      /* Copy the row received into its place in the image */
169      memcpy(&A(num_row, 0), B, w);
170    }
171  } else {/* CODE FOR WORKERS */
172    /* Receive the number of the row to be computed, or an out-of-range number to end */
173    MPI_Recv(&num_row, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
174    while ( num_row < h ) {
175      /* Compute the row */
176      for ( j = 0 ; j < w ; j++ ) {
177        z0 = (x1 + ix*j) + (y1 + iy*num_row)*I;
178        ni = newton(z0, tol, maxiter);
179        if ( ni > max ) max = ni;
180        B[j] = ni;
181      }
182      /* Send the computed row */
183      MPI_Send(B, w, MPI_BYTE, 0, num_row, MPI_COMM_WORLD);
184      /* Receive the number of the next row to be computed */
185      MPI_Recv(&num_row, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
186    }
187
188  }
```

El trabajador procesa la fila **num_row**

- **next_row(Maestro)**:
  Índice de la fila que va a ser procesada
- **rows_done (maestro)**:
  Número de filas que se han procesado
- **proc (maestro)**:
  Índice del proceso que le ha enviado la fila procesada
- **num_row**:
  - **Maestro**:
    Índice de fila recibida (procesada)
  - **Trabajador**:
    Índice de fila que a procesar el trabajador
- h=número de filas de la imagen
- w= número de columnas de la imagen

- El array **A** se usa en el proceso maestro para guardar la imagen.
- **A** es un array unidimensional, pero hay definida al principio del código una macro que permite usarlo como una matriz:
  #define A(i, j) A[(i)*w + (j)]

Ejercicios 3 y 4

- Modifica la implementación newton.c, llamándola por ejemplo newtonm.c, para que siga el esquema maestro-trabajador en donde el maestro también trabaja (solo se modifica la parte del maestro)
  - Sustituye el código correspondiente a la figura 4 por el correspondiente de la figura 5
  - Prueba el código implementado y comprueba que se obtienen los mismos fractales para las opciones –c1 a –c4.

```
/* Recibimos una fila ya calculada de cualquier proceso */
MPI_Recv(B, w, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
         &status);
```

Figura 4

```
siguiente_fila <- 0
Para proc = 1 ... np
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
fin_para

filas_hechas <- 0
Mientras filas_hechas < filas_totales
  recibe de cualquier proceso una fila calculada
  proc <- proceso que ha enviado el mensaje
  num_fila <- número de fila
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
  copia fila calculada a su sitio, que es la fila num_fila de la imagen
  filas_hechas <- filas_hechas + 1
fin_mientras
```

Figura 5

```
siguiente_fila <- 0
Para proc = 1 ... np
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
fin_para

filas_hechas <- 0
Mientras filas_hechas < filas_totales
  inicia la recepción no bloqueante de una fila calculada de cualquier proceso
  Mientras no se ha recibido nada y siguiente_fila < filas_totales
    procesa la fila número siguiente_fila
    siguiente_fila <- siguiente_fila + 1
    filas_hechas <- filas_hechas + 1
  fin_mientras
  Si no se ha recibido nada entonces
    espera (de forma bloqueante) a recibir algo
  fin_si
  proc <- proceso que ha enviado el mensaje
  num_fila <- número de fila
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
  copia fila calculada a su sitio, que es la fila num_fila de la imagen
  filas_hechas <- filas_hechas + 1
fin_mientras
```

Código distinto

Ejercicios 3 y 4

```
if (me == 0) {
  /* CODE FOR THE MASTER */
  /* Initial distribution of work */
  next_row = 0;
  for ( proc = 1 ; proc < np ; proc++ ) {
    MPI_Send(&next_row, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
    next_row++;
  }
  /* While there are rows to be received */
  for ( rows_done = 0 ; rows_done < h ; rows_done++ ) {
    /* Receive a computed row from any process */
    MPI_Recv(B, w, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
             &status);
    /* Get the process index and the row number */
    proc = status.MPI_SOURCE;
    /* The row number is in the message TAG */
    num_row = status.MPI_TAG;
    /* Ask that process to compute another row */
    MPI_Send(&next_row, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
    next_row++;
    /* Copy the row received into its place in the image */
    memcpy(&A(num_row, 0), B, w);
  }
} else {
  /* CODE FOR WORKERS */
  /* Receive the number of the row to be computed, or an out-of-range number to end */
  MPI_Recv(&num_row, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  while ( num_row < h ) {
    /* Compute the row */
    for ( j = 0 ; j < w ; j++ ) {
      z0 = (x1 + ix*j) + (y1 + iy*num_row)*I;
      ni = newton(z0, tol, maxiter);
      if ( ni > max ) max = ni;
      B[j] = ni;
    }
    /* Send the computed row */
    MPI_Send(B, w, MPI_BYTE, 0, num_row, MPI_COMM_WORLD);
    /* Receive the number of the next row to be computed */
    MPI_Recv(&num_row, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
}
```

sustituir

```
while (MPI_Test(&request, &msg_received, &status),!msg_received && next_row<h)
```

```
inicia la recepción no bloqueante de una fila calculada de cualquier proceso
Mientras no se ha recibido nada y siguiente_fila < filas_totales
  procesa la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
  filas_hechas <- filas_hechas + 1
fin_mientras
Si no se ha recibido nada entonces
  espera (de forma bloqueante) a recibir algo
Fin_si
```

Ejercicios 3 y 4

```
siguiente_fila <- 0
Para proc = 1 ... np
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
fin_para

filas_hechas <- 0
Mientras filas_hechas < filas_totales
  inicia la recepción no bloqueante de una fila calculada de cualquier proceso
  Mientras no se ha recibido nada y siguiente_fila < filas_totales
    procesa la fila número siguiente_fila
    siguiente_fila <- siguiente_fila + 1
    filas_hechas <- filas_hechas + 1
  fin_mientras
  Si no se ha recibido nada entonces
    espera (de forma bloqueante) a recibir algo
  fin_si
  proc <- proceso que ha enviado el mensaje
  num_fila <- número de fila
  envía al proceso proc petición de hacer la fila número siguiente_fila
  siguiente_fila <- siguiente_fila + 1
  copia fila calculada a su sitio, que es la fila num_fila de la imagen
  filas_hechas <- filas_hechas + 1
fin_mientras
```

Figura 5: Pseudo-código del maestro haciendo que también trabaje (el código de los trabajadores no cambia).

Procesamiento de la fila de los trabajadores

```
for ( j = 0 ; j < w ; j++ ) {
      z0 = (x1 + ix*j) + (y1 + iy*num_row)*I;
      ni = newton(z0, tol, maxiter);
      if ( ni > max ) max = ni;
      B[j] = ni;
    }
```

- Debes tener en cuenta que el índice de la fila que procesan los trabajadores es num_row; sin embargo, el índice de fila que tiene que procesar el maestro es next_row
- Fíjate también que los trabajadores almacenan en el vector B la fila procesada; sin embargo, el proceso maestro debe almacenar directamente la fila procesada en la matriz A. Para hacerlo, debes tener en cuenta la macro de la línea 133, para almacenar la fila de índice next_row en A.

```
133    #define A(i, j) A[(i)*w + (j)]
```

Recepción no bloqueante en MPI

MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, **MPI Request *request**)

request: puntero a la variable que almacena la petición no bloqueante

Es necesario asegurarse antes de modificar la variable que va a contener el mensaje esperar o comprobar que se ha realizado el envío del mensaje:

- Espera:

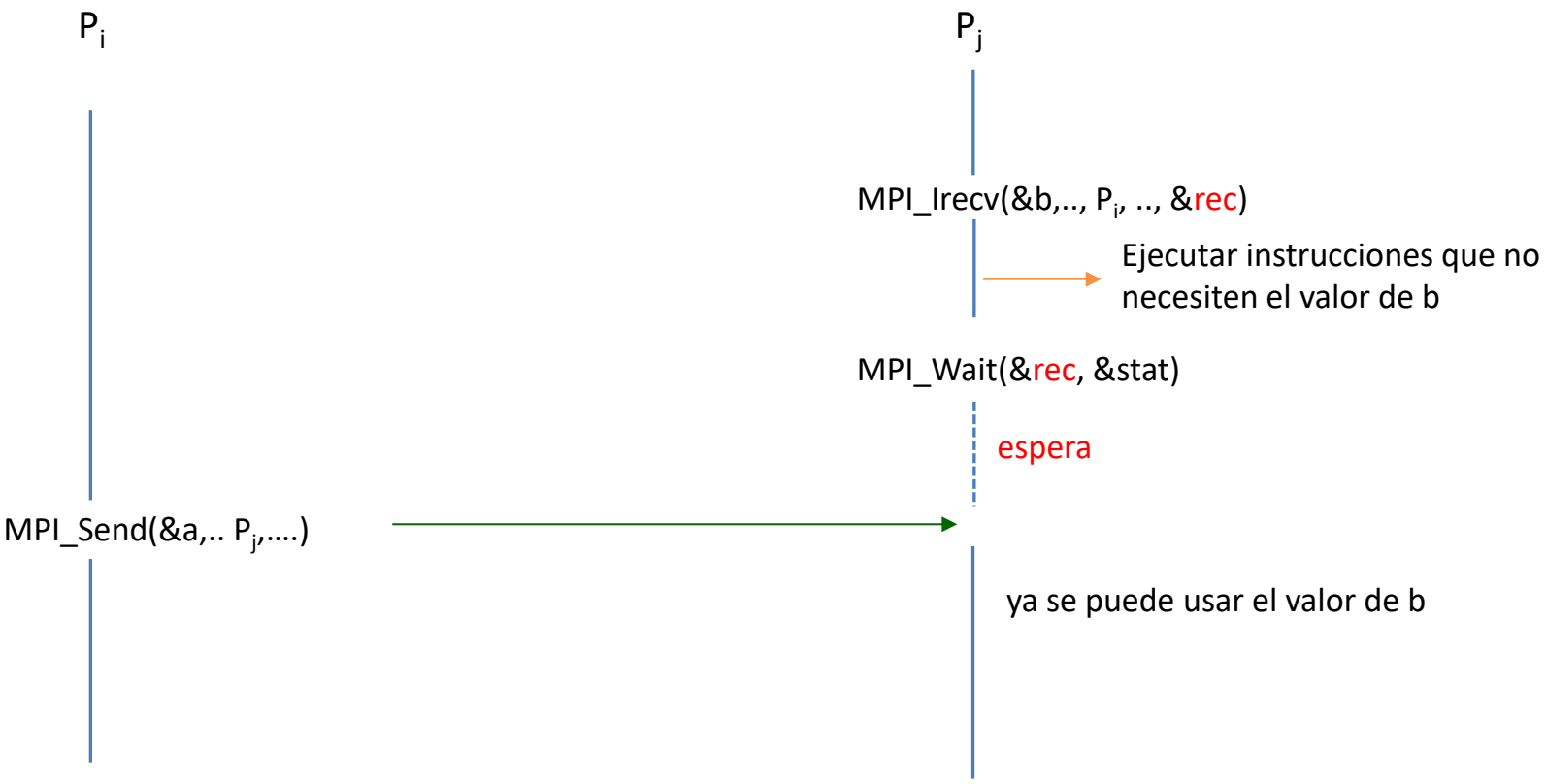
  MPI_Wait(MPI_Request *request, MPI_Status *status)

- Comprobación:

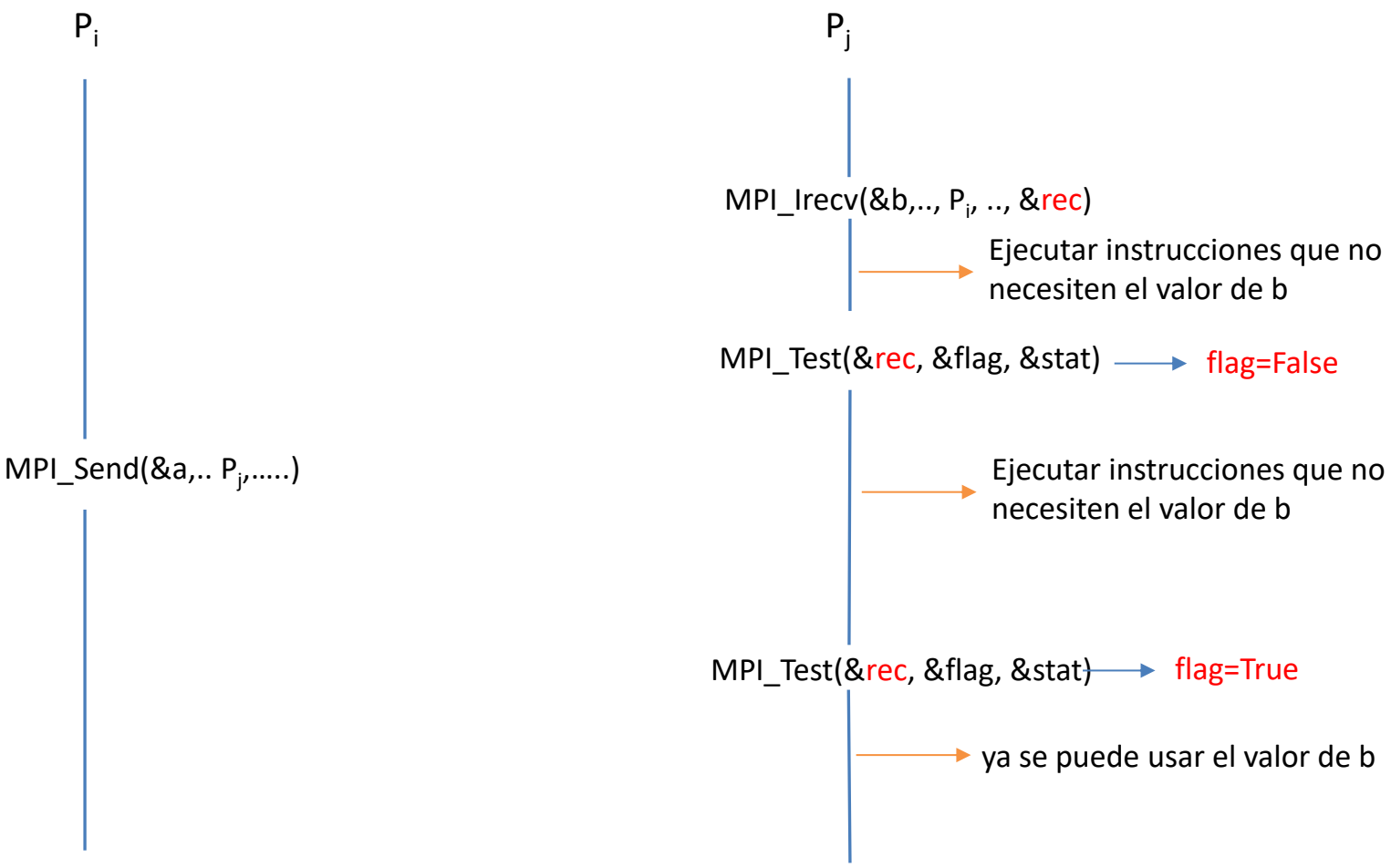
  MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)

  - **flag**: 0 si no se ha producido la petición de recepción del mensaje, 1 si se ha producido
  - **status**: puntero al estado de la comunicación

Recepción no bloqueante (a y b datos simples)



Recepción no bloqueante (a y b datos simples)



Análisis prestaciones

- Calcula tiempos de ejecución en Kahan para los dos códigos usando la opción –c5

Script

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks=2
#SBATCH --time=5:00
#SBATCH --partition=cpa
scontrol show hostnames $SLURM_JOB_NODELIST
echo maestro no colabora
mpiexec newton –c4
echo maestro colabora
mpiexec newtons
```

Resultados:

```
kahan02
kahan03
maestro no colabora
Tiempo: 22.37 (2 procesos)
maestro colabora
Tiempo: 13.41 (2 procesos)
```