Práctica 3

Sesión 2

Objetivos

- 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), que se encuentra en las zona de tareas de POLIFORMAT, para calcular fractales de Newton, que habrá que modificar convenientemente.

Maestro no colabora  Maestro colabora

Fractales (II)

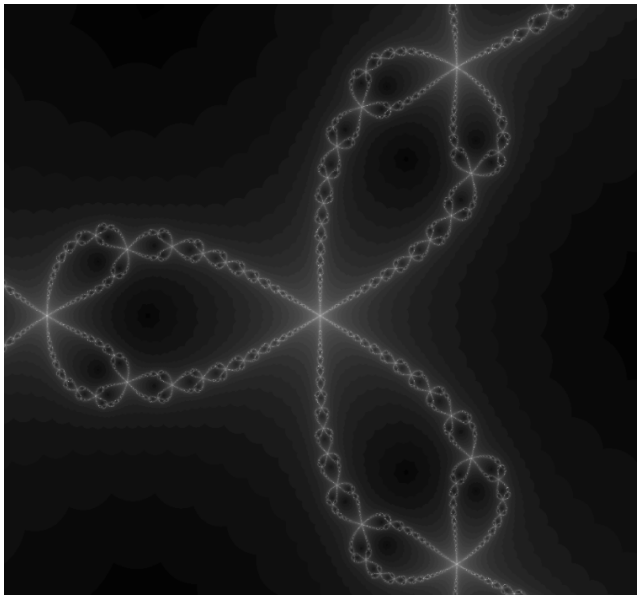


Fig 3. $f(z)=z^3-1$.

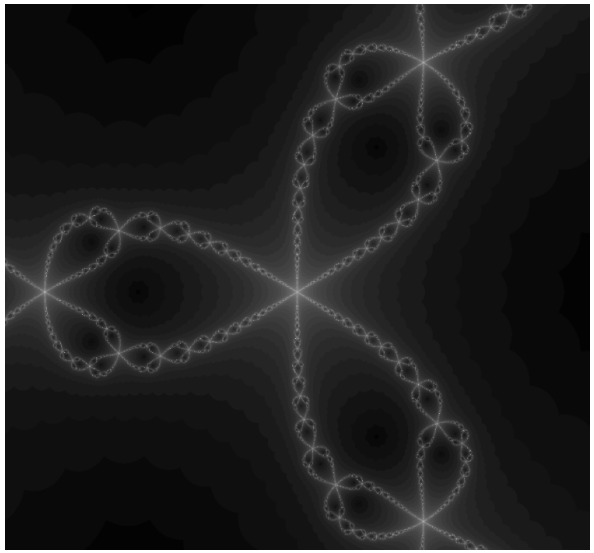
```
Algoritmo
Para y = y1,..., y2
    Para x = x1,..., x2
        col = num_iteraciones_Newton(funcion,x,y)
        Pintar el punto (x,y) con el color col
    fin_para
fin_para
```

- Para dibujar el color en un punto (x_0,y_0), se calcula el número de iteraciones necesarias para que el método iterativo de Newton converja.
- El método de Newton es un método iterativo que permite obtener una raíz compleja de la ecuación $f(x)=0$, partiendo de un punto (x_0,y_0).
- El máximo número de iteraciones se usa para el color blanco.

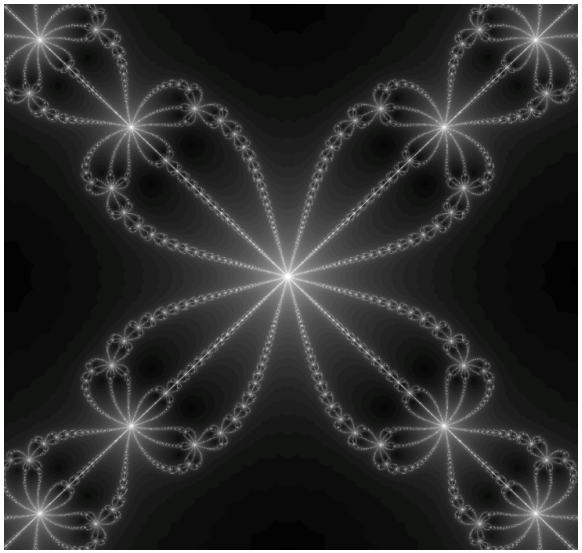
Ejercicio 1

- Compila y ejecuta el programa newton.c, para los 5 fractales básicos que puede generar: para ello utiliza las opciones -c1, -c2, -c3, -c4 y -c5 (cada opción genera un fractal a partir de una determinada función)
  - La opción –c5 corresponde a un fractal “no muy bonito”, pero ideal para analizar las prestaciones de las implementaciones paralelas por ser más costoso que el resto.
  - Prueba con la opción -p50 o con -p-11 para colorear.
- El programa genera una imagen en escala de grises .pgm o .ppm (o color, con la opción −p num). “num” permite variar la paleta de colores.
- El nombre de la imagen se puede cambiar con la opción −o.

Fractales opciones –c1 a –c4



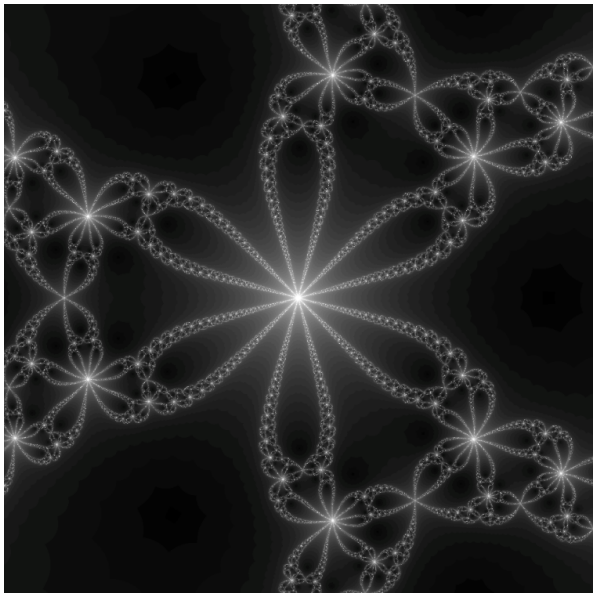
-C1



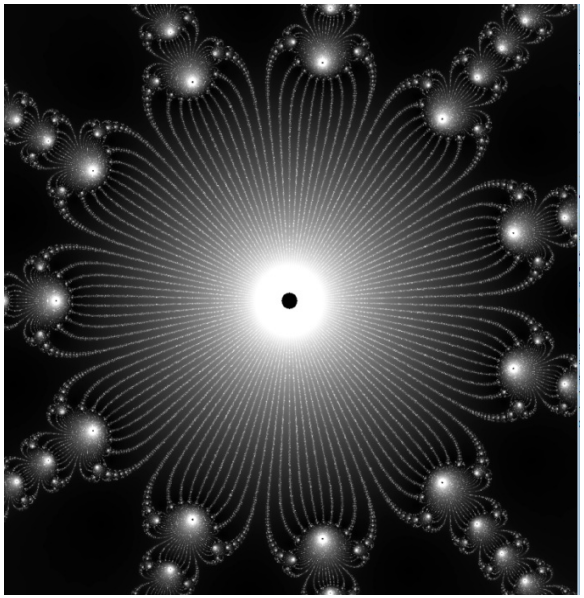
-C2

Nota:

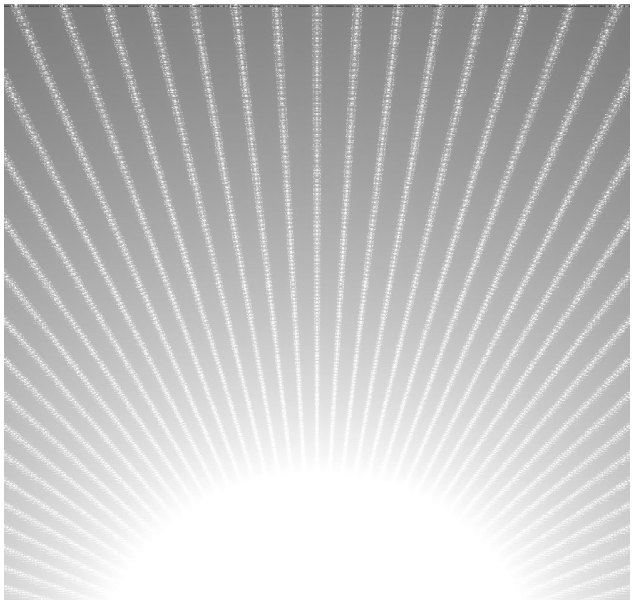
- Los puntos cuyo cálculo tiene un mayor coste computacional son los puntos en blanco
- -C5 es el fractal más costoso



-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
        envía al proceso proc petición de calcular 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 <- numero 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
si_no (soy un trabajador)
    recibe número de fila a hacer en num_fila
    Mientras num_fila < filas_totales
        procesa la fila número num_fila
        envía la fila recien calculada al maestro
        recibe número de fila a hacer en num_fila
    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

```
if(yo == 0) {/* CÓDIGO DEL MAESTRO */
  /* Reparto inicial de faena */
  siguiente_fila = 0;
  for ( proc = 1 ; proc < np ; proc++ ) {
    MPI_Send(&siguiente_fila, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
    siguiente_fila++;
  }
  /* Mientras queden filas por recibir */
  for ( filas_hechas = 0 ; filas_hechas < h ; filas_hechas++ ) {
    /* Recibimos una fila ya calculada de cualquier proceso */
    MPI_Recv(B, w, MPI_BYTE, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD,
             &status);
    /* Sacamos el identificador del proceso y el número de fila */
    proc = status.MPI_SOURCE;
    /*  num_fila contiene el número de fila recibido*/
    num_fila = status.MPI_TAG;
    /* Le pedimos a ese proceso que procese otra fila */
    MPI_Send(&siguiente_fila, 1, MPI_INT, proc, 0, MPI_COMM_WORLD);
    siguiente_fila++;
    /* Copiamos la fila procesada a su lugar en la imagen */
    memcpy(&A(num_fila, 0), B, w);
  }
}else{/* CÓDIGO DE LOS TRABAJADORES */
  /* Recibe número de fila con la que trabajar, fuera de rango para acabar */
  MPI_Recv(&num_fila, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  while ( num_fila < h ) {
    /* Calcula esa fila */
    z0.b = y1 + iy*num_fila;
    for ( j = 0 ; j < w ; j++ ) {
      z0.a = x1 + ix*j;
      ni = newton(z0, tol, maxiter);
      if ( ni > max ) max = ni;
      B[j] = ni;
    }
    /* Envía fila recién calculada */
    MPI_Send(B, w, MPI_BYTE, 0, num_fila, MPI_COMM_WORLD);
    /* Recibe número de la siguiente fila a procesar */
    MPI_Recv(&num_fila, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
  }
}
```

- **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

Fig 5. Algoritmo maestro-trabajador clásico (maestro colabora)

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 <- numero 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

Sustituir está línea por el código del recuadro verde

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

Ejercicios 3 y 4

- Modifica la implementación newton.c, llamándola por ejemplo newtoncm.c (o newtonsol.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 al recuadro la izquierda por el correspondiente de la derecha
  - Prueba el código implementado y comprueba que se obtienen los mismos fractales para las opciones –c1 a –c4.
  - Calcula tiempos de ejecución para los dos códigos usando la opción –c5 y calcula los speedups y eficiencias (considera la ejecución secuencial aquella en la que hay un solo trabajador

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

Ejercicio 3

```
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).

```
si_no (soy un trabajador)

  recibe número de fila a hacer en num_fila
  Mientras num_fila < filas_totales
    procesa la fila número num_fila
    envía la fila recién calculada al maestro
    recibe número de fila a hacer en num_fila
  fin_mientras
```

```
/* Calcula esa fila */
z0.b = y1 + iy*num_fila;
for ( j = 0 ; j < w ; j++ ) {
  z0.a = x1 + ix*j;
  ni = newton(z0, tol, maxiter);
  if ( ni > max ) max = ni;
  B[j] = ni;
}
```

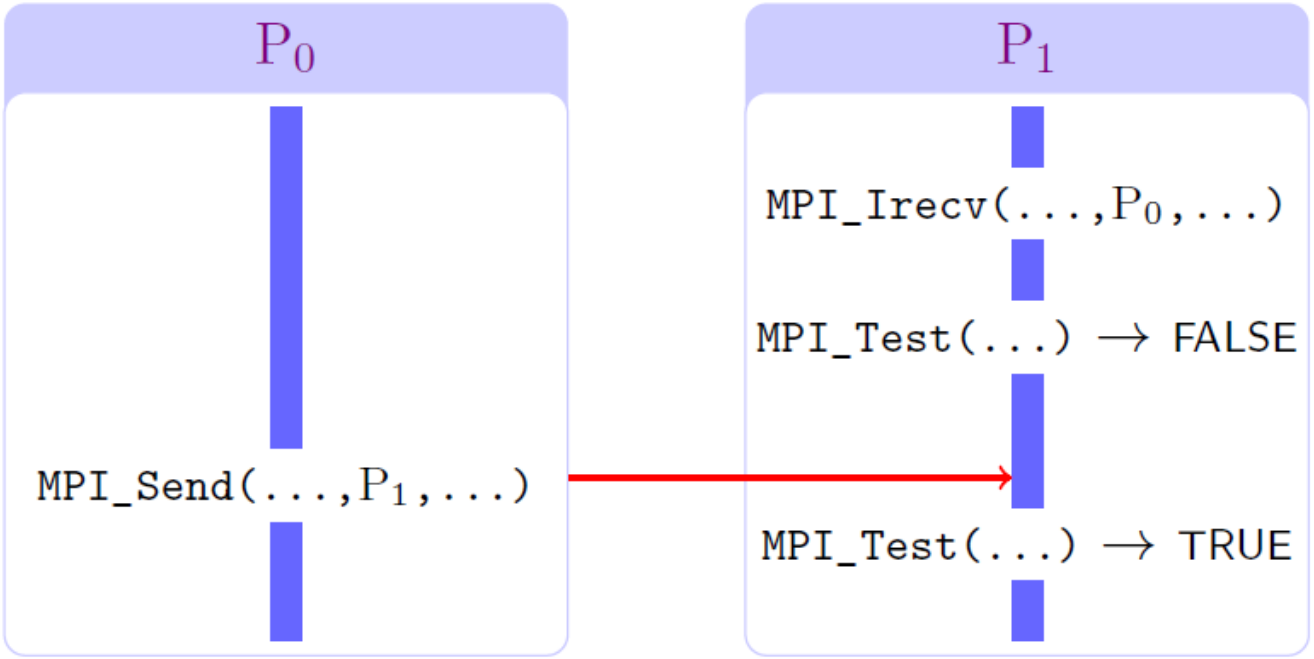
El maestro tiene que procesar la siguiente fila: debes adaptar el procesamiento de la fila que hacen los esclavos al procesamiento de fila que tiene que hacer el maestro, copiando la fila calculada en su sitio

Recepción No Bloqueante

```
MPI_Irecv(buf, count, type, src, tag, comm, req)
```

Se inicia la recepción, pero el receptor no se bloquea

- Tiene un argumento adicional (`req`)
- Es necesario comprobar después si el mensaje ha llegado



- Ventaja: solapamiento de comunicación y cálculo
- Inconveniente: programación más difícil

Recordatorio funciones MPI_Irecv, MPI_Wait, MPI_Test

`int` **`MPI_Irecv`**`(void *buf, int count, MPI_Datatype datatype, int source, int tag,MPI_Comm comm, MPI_Request *request)` → Recepción sin bloqueo

`int` **`MPI_Wait`**`(MPI_Request *request, MPI_Status *status)` → Espera a recepción mensaje

`int` **`MPI_Test`**`(MPI_Request *request, int *flag, MPI_Status *status)` → Comprueba recepción de mensaje

- `request:` identificador de la recepción no bloqueante
- `status:` estado del mensaje recibido
  - status.MPI_SOURCE: identificador del proceso que ha realizado el envío
  - status.MPI_TAG: etiqueta del mensaje recibido
- `flag:`
  - 1 El emisor ha realizado el envío (éxito)
  - 0 El emisor no ha realizado el envío (aún)