Práctica 3: Paralelización con MPI
Sesión 2: Fractales de Newton

Mónica Chillarón

Computación Paralela (CPA)

Curso 2020/2021

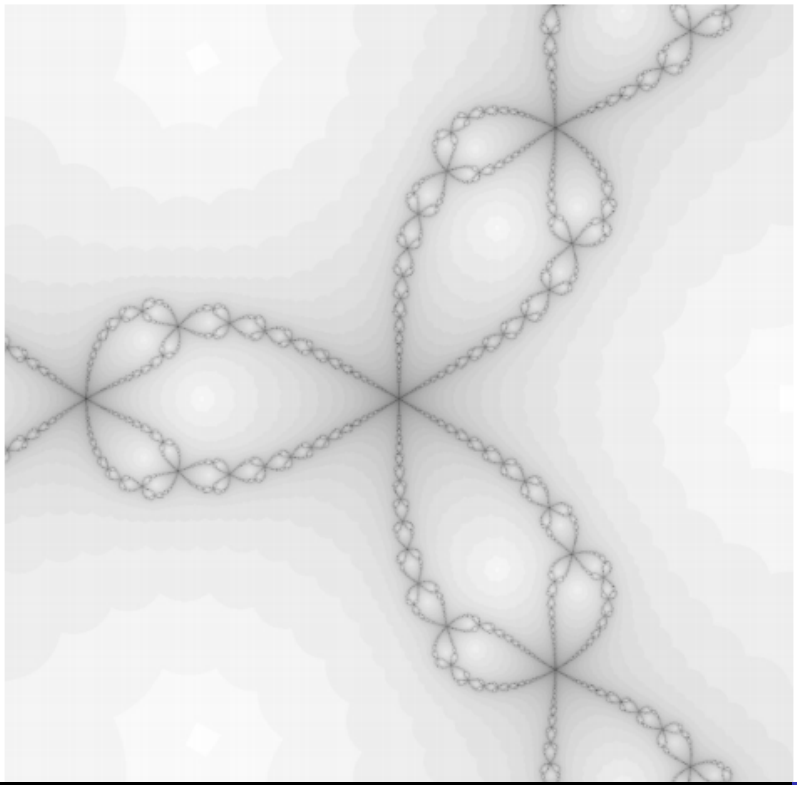
DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

Algoritmo Secuencial

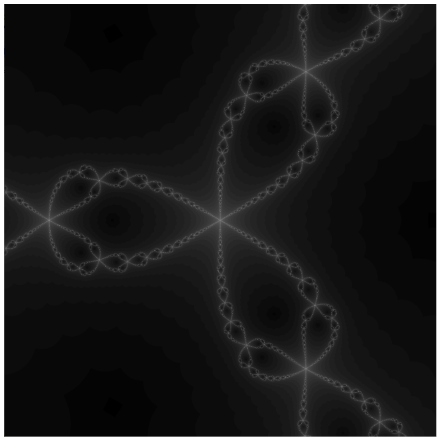
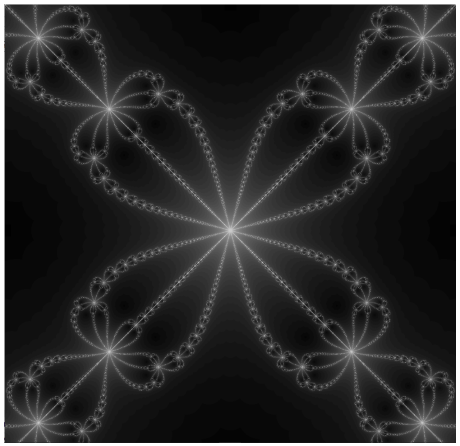
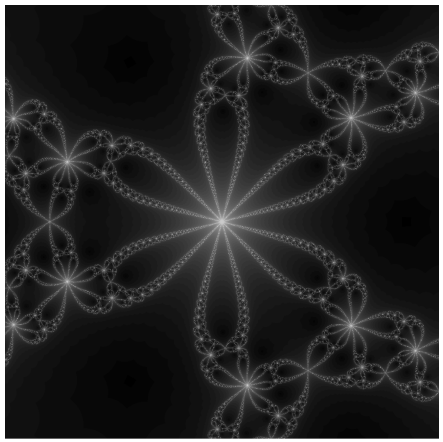
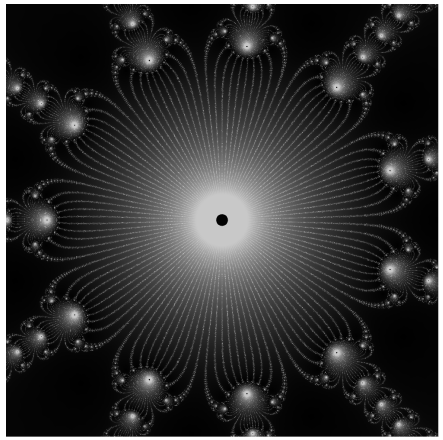
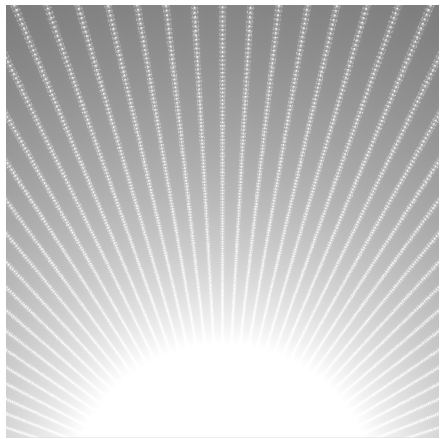
- Dibujo de fractales mediante método de Newton
- Método iterativo que busca raices complejas de una función desde un punto (x,y) inicial
- Utilizaremos las iteraciones que haya tardado en converger como color del píxel (x,y)
- El color blanco equivale al número máximo de iteraciones



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

Algoritmo Secuencial

Hay definidas 5 funciones que se pueden escoger mediante **-c [1-5]** al llamar al programa:

Algoritmo Maestro-Trabajadores Clásico

```
Si yo=0 entonces (soy el maestro)
```

```
siguiente_fila <- 0
Para proc = 1 ... np-1
  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
```

```c
if (me == 0) {

  /* CODE FOR THE MASTER */
```

```c
  /* 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++;
  }
```

```c
  /* 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);
  }
```

Algoritmo Maestro-Trabajadores Clásico

```
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

Fin_si
```

```
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 */
    z0.b = y1 + iy*num_row;
    for ( j = 0 ; j < w ; j++ ) {
      z0.a = x1 + ix*j;
      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);
  }

}
```

Algoritmo Maestro-Trabajadores Clásico

- Ejercicio 1: Compila y ejecuta el programa usando el caso 1. Haz una copia de la imagen obtenida (newton.pgm) con el nombre ref.pgm. Este fichero te servirá para comprobar que la modificación que realices al programa funciona correctamente.

- Ejercicio 2: Repasa el algoritmo utilizado (Figura 4 del boletín) y su implementación en lenguaje C realizada en la función fractal_newton del programa proporcionado. Responde a las siguientes preguntas:
  1. Cuando un proceso trabajador le envía una fila al proceso maestro, ¿cómo le indica de qué fila se trata?
  2. ¿Cómo sabe un proceso trabajador que ya no tiene que procesar más filas?

Algoritmo Maestro-Trabajadores con el Maestro trabajando

- Si ejecutamos reservando un procesador para que sólo sea maestro, estaremos desaprovechando la potencia de cálculo de este procesador.
- Una forma de no desaprovechar esta potencia extra es haciendo que el proceso maestro también realice trabajos.
- Para ello, hay que hacer que sus comunicaciones sean no bloqueantes. De esta manera, en lugar de quedarse bloqueado esperando la respuesta de algún trabajador, podrá ir trabajando al mismo tiempo que espera esta respuesta.

Algoritmo Maestro-Trabajadores con el Maestro trabajando

```
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
```

Replicaremos el código que utilizan los procesos trabajadores para calcular una fila

- Este resultado se escribirá directamente en el array A mediante la macro:

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

en lugar de utilizar el array B como hacen los trabajadores

Algoritmo Maestro-Trabajadores con el Maestro trabajando

Recepciones no bloqueantes:

- MPI_Irecv(void *buf, int count,MPI_Datatype datatype, int source, int tag,MPI_Comm comm, MPI_Request *request)
- MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)
- MPI_Wait(MPI_Request *request,MPI_Status *status)

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

Algoritmo Maestro-Trabajadores con el Maestro trabajando

- Ejercicio 3: Lee y entiende el algoritmo mostrado en la Figura 5 del boletín e impleméntalo sobre una copia del programa original, para luego poder comparar ambos programas.

- Ejercicio 4: Utiliza algún fractal más costoso (el caso 5, por ejemplo) para sacar medida de prestaciones comparando los resultados de ambos programas. Ejecuta ambos programas, por ejemplo con 4 y con 8 procesos, y comprueba cuál de las dos versiones funciona mejor.