Introducción

Un fractal es un objeto geométrico cuya estructura básica, fragmentada o aparentemente irregular, se repite a diferentes escalas. El conjunto de Mandelbrot es uno de los fractales más estudiados, se define en el plano complejo y se construye a partir de una sucesión,

$$\begin{cases} z_0 & = 0 \in \mathbb{C} \quad \text{punto inicial,} \\ z_{n+1} & = z_n^2 + c \quad \text{sucesión recursiva.} \end{cases} \tag{1}$$

Para este trabajo, se generó un vector de n x m, y se buscó si cada elemento del vector pertenecía o no al conjunto de Mandelbrot. Después, se generó una imagen de tamaño (n, m) tipo pgm para su visualización. Se compararon los tiempos de ejecución en secuencial y en paralelo, utilizando el modelo maestro-trabajador para este último.

Metodología

Función Mandelbrot

La función de Mandelbrot se define como aquella que dado un punto complejo p, mientras p no sea mayor a una norma en un máximo de iteraciones, el punto p pertenece al conjunto, en otro caso, no pertenece.

Algoritmo 1 Mandelbrot

Entrada: Punto complejo p = a + bi, máximo de iteraciones (MAX_ITER), máximo tamaño de norma (MAX_NORM), punto inicial z_0 = c + di.
Salida: 1 si pertenece al conjunto y 0 si no.

```
for i in MAX_ITER do
    z_real = c*c - d*d + a,
    z_imag = 2*c*d + b,
    if norma (z) > MAX_NORM then
        return 0
    else
        z_0 = z.
return 1.
```

Código secuencial

El eje x se definió como el eje real y el eje y como el complejo.

Algoritmo 2 Mandelbrot Secuencial

Entrada: Número de puntos: numPix, dominio en x = [x_begin, x_end], dominio en y = [y_begin, y_end], tamaño de paso en x (xsize), tamaño de paso en y (ysize).
Salida: Vector binario de tamaño numPix.

```
Inicializar vector result de tamaño numPix,
for i in numPix do
    p_real = x_begin + (i%n)*xsize,
    p_imag = y_begin + (i/m)*ysize,
    result[i] = mandelbrot(p).
```

Código paralelo

El modelo maestro-trabajador implica que el proceso maestro manda instrucciones a los procesos trabajadores, y cuando los trabajadores tienen resultados, se los envían al proceso maestro. Una forma de paralelizar el código secuencial era partir el tamaño del vector entre el número de procesos menos uno, de esta forma, el

maestro envía los indices de inicio y fin a cada proceso, y recibe los resultados para después hacer la imagen. Sin embargo, para aprovechar que hay puntos que son más rápidos que otros, primero se tenía que revolver los puntos y mandarlos sin orden para cada proceso trabajador.

Sin embargo, también se podían encontrar muchos puntos dentro del conjunto y podría ocurrir que un proceso siga trabajando y que otros hayan terminado. Otra forma de hacerlo (y fue la que se hizo en este trabajo), fue que dados n procesos trabajadores, a cada proceso se le mandó los primeros n pixeles. Después, cuando fueran acabando se mandaban los siguientes pixeles sin orden, es decir, si se busca mandar 10 pixeles = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] y se tienen tres procesos, primero a cada proceso i, le corresponde el pixel 1, 2, 3. Si el proceso 3 fue el primero en terminar, este recibe el siguiente pixel 4; si el siguiente fue el pixel 1, este recibe el pixel 5, y así sucesivamente.

Algoritmo 3 Mandelbrot Paralelo

Entrada:
Salida:

```
1:  Inicializar entorno MPI
2:  obtener número de procesos y cantidad de procesos.
3:  function RUN PARALLEL
4:      idx_pix = 0
5:      if estamos en el proceso maestro then
6:          for i en los procesos trabajadores do
7:              idx_pix += 1,
8:              Enviar pixeles: MPI_Send(idx_pix)
9:          while haya pixeles por calcular do
10:             Recibir resultado de pixel del proceso trabajador: MPI_Recv(idx_pix, resultado)
11:             Actualizar vector de resultados,
12:             Obtener el proceso que ya terminó con la función "status.MPI_SOURCE",
13:             if Hay pixeles por calcular then
14:                 Enviar pixeles restantes: MPI_Send(pixeles faltantes)
15:             Generar la imagen.
16:             Mandar mensaje de que ya no hay pixeles.
17:             Limpiar memoria.
18:     else
19:         Recibir indice: MPI_Recv(idx_pix),
20:         Calcular si pertenece o no al conjunto.
21:         Enviar al proceso maestro: MPI_Send(idx_pix, resultado).
22: Terminar entorno MPI.
```

Resultados

Se compararon los tiempos de ejecución del código secuencial y paralelo con cinco iteraciones, en el Cuadro 1. se muestran los resultados. Además, en la Fig. 1 se muestra que los resultados obtenidos con ambos procedimientos, obtuvieron los mismo resultados.

	Promedio	**Desviación estándar**	**Mínimo**	**Mediana**	**Máximo**
Secuencial	2621.182	1.4607	2619.56	2620.98	2623.21
Paralelo	240.6782	1.9429	239.229	239.642	243.933

Cuadro 1: Resultados de ejecuciones en segundos. Para el resultado en paralelo, se promedio cada iteración de los 12 procesos.

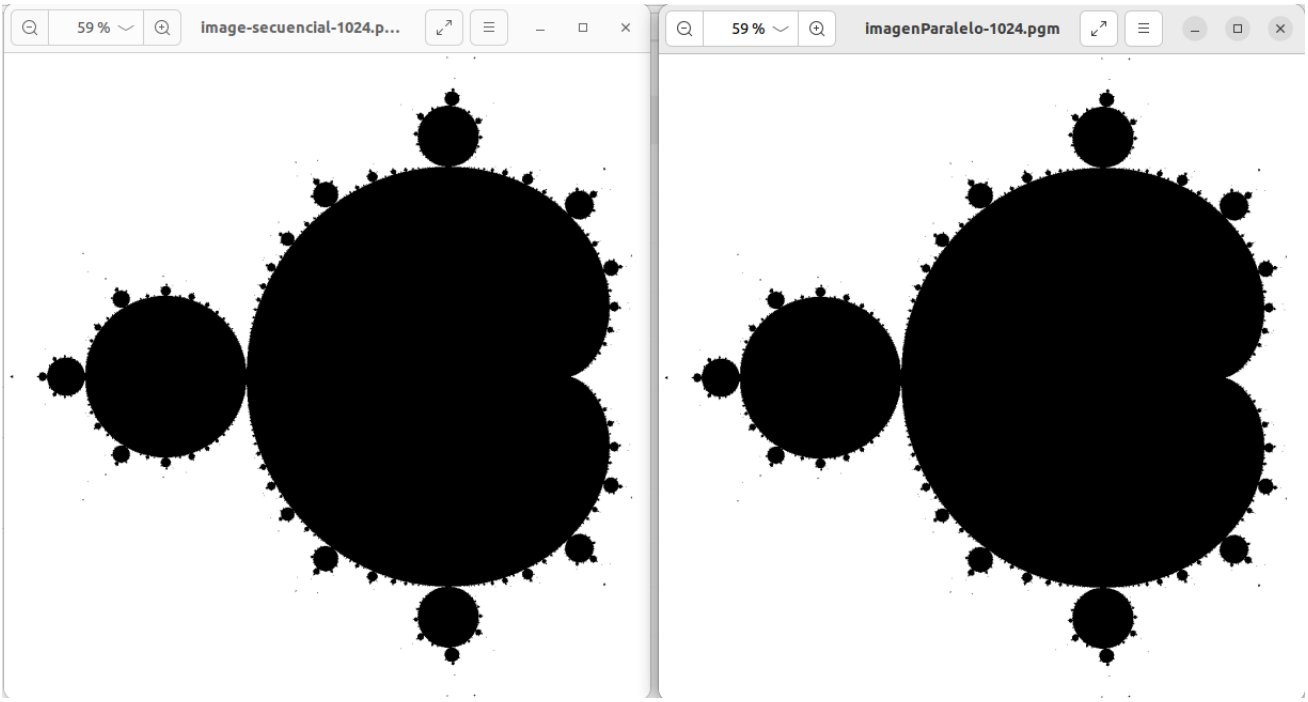

Figura 1: Resultados en imagen pgm, en secuencial y en paralelo.

Conclusiones

Se encontró que el procedimiento en paralelo disminuyó de forma significativa el tiempo y que se obtuvieron los mismos resultados.