Computer Architecture

OpenMP assignment
Mandelbrot set

Serafin.Benito@uclm.es

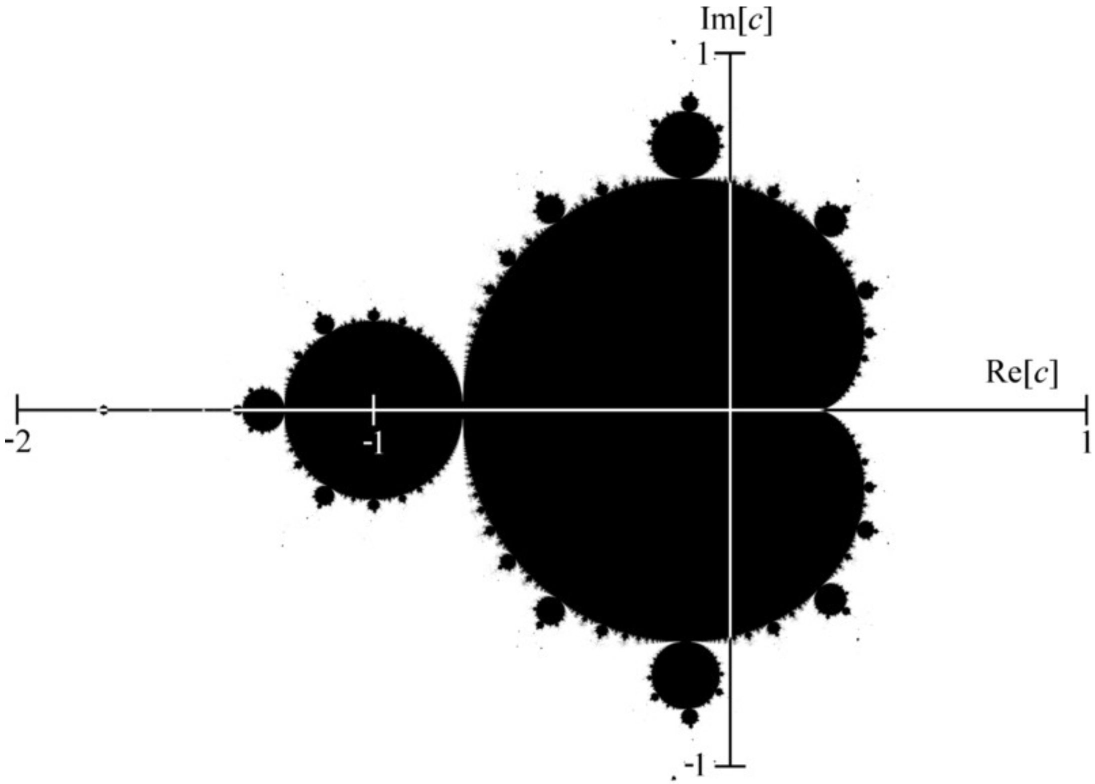
Contents

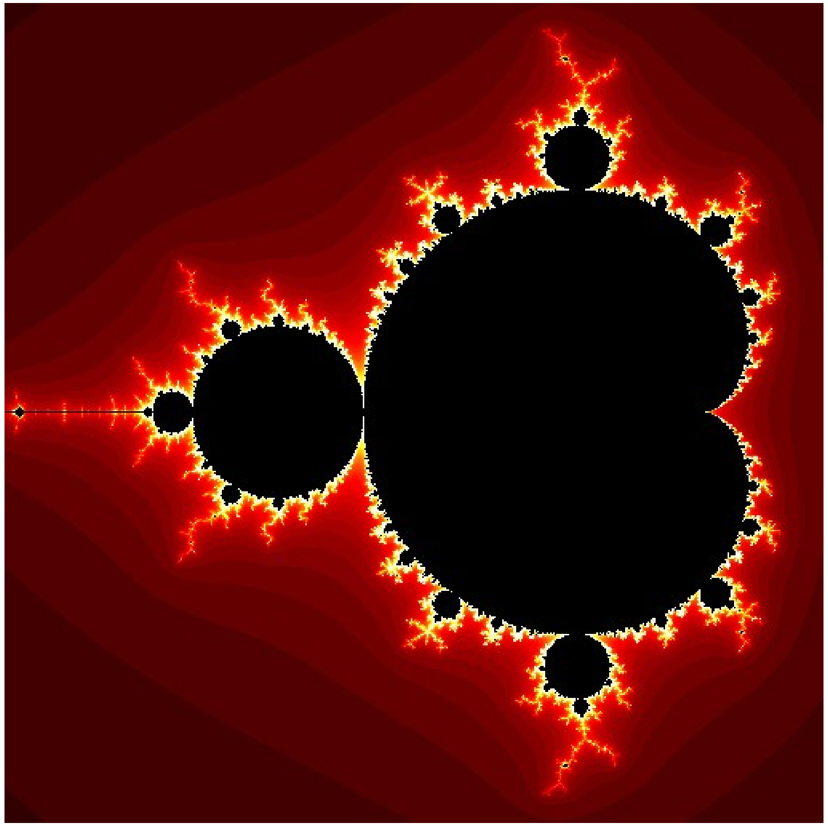
- Mandelbrot Set
- Programming Mandelbrot
- Test of the program
- Parallelization with OpenMP
- Execution times
- Handout

Mandelbrot Set

Best known of fractals

Represented in the complex plane: number **x+i·y** corresponds to the coordinate point **(x,y)**





Set definition

- Let's call M to the Mandelbrot Set
- Given a complex number *c*, we build the sequence defined by:

$$z_0 = 0$$
$$z_{n+1} = z_n^2 + c$$

- $c \in M$ if and only if the sequence is bounded. Ex.:
  - $c = -1 + i \rightarrow 0, -1+i, -i, -1+i, -i, -1+i \ldots$ bounded
  - $c = -1 \rightarrow 0, -1, 0, -1 \ldots$ bounded
  - $c = 1 \rightarrow 0, 1, 2, 5, 26, 677 \ldots$ not
- Thus, $-1+i \in M, -1 \in M$ and $1 \notin M$

Programming Mandelbrot

- There’s no know method that, $\forall c$,
  tells if $c \in M$ or $c \notin M$
  - All representations of the Mandelbrot set are approximate
- We know that $c \in M \;\; iff \;\; \forall n \in \mathbb{N} \;\; |z_n| \le 2$
- The algorithm computes (from *c*) the terms of the sequence up to a maximum of terms:
  - If it founds a term $z_n \, where \, |z_n| > 2, \;\; c \notin M$
  - Otherwise, it considers $c \in M$ although it may not be (maybe in the rest of the sequence there’s a term with modulo >2)

Input file

- Execution:

<executable> <input file>

- The input file is a text file with the following format:
  - 1st line: number of images to generate
  - An additional line per every image. Possible line formats:
    - 1 <minor abscissa> <major abscissa> <minor ordinate> <major ordinate> <image file name>
    - 2 <abcissa center of square> <ordinate center> <square size> <image file name>
    - The image file name can't have more than 15 characters nor a dot

Sequential program

A. Open the input file and read the number of images
B. For every image
   - Read image type (1 = rectangular, 2 = square)

   B1. Compute, in pixels, witdh and height of the image

   B2. Open and configure ppm image file

   B3. For every pixel compute coordinate (x,y) and call function **mandel_val** that determines if it can belong to the Mandelbrot set or not

   B4. Compute the color of the pixel that corresponds to number $x + y \cdot i$ and print it in the image file

   B5. Print the time taken to obtain the image

Function mandel_val

- Parameters list: (x,y,max_iter)
- From $c = x + i \cdot y$, it generates the terms of its sequence (`p_real + i·p_imag`) up to a a maximum of max_iter-1 terms:
  - If a term whose module is >2 is found, the loop is broken and index j of that term of the sequence returned
    - The color of the pixel will depend on the number j of iterations
  - Otherwise, it returns -1 (number $x + i \cdot y$ will be drawn in white, as belonging to the Mandelbrot Set)

Test of the program

- Create the input file. With the following content, for example:

  2
  2 -0.2 .8 .05 cuadrado
  1 -2 1 -1 1 rectangulo

  It will create 2 images (1st line):
  - First square (2 in the beginning of 2nd line), in file cuadrado.ppm centered in (-0.2, 0.8) and a square of size 0.05
  - Second rectangular (1 in the beginning of 3rd line), in file rectangulo.ppm, from abscissa -2 to abscissa 1, and ordinates from -1 to 1
- Compile and run the program as described in the initial comment lines
- Try to change the input file an test cocrete zones
  - The smallest the area explored, the biggest the ***zoom***
  - The most interesting zones are at the borders

Parallelization with OpenMP

- Build a **`mandelbrot_paralelo.c`** as a result of the parallelization of the sequential version with OpenMP (`mandelbrot_secuencial.c`)
- The computations of the sequences that correspond to the complex numbers of the zone to explore are independent →

  → the can be performed **in parallel**

Parallelization with OpenMP

- At the time of parallelizing, the order in which results are generated is different from the sequential execution
- But **pixels must be printed** in the image file (that we'll call **`mandelbrot_paralelo.ppm`**) **in** the same **order** used by the **sequential** algorithm.
- Suggestion: store the generated results in a matrix and, when complete, use the matrix to print the pixels
  - Points B4 and B5 of the algorithm, now must be solved separately

Execution times

- Using the same input file, run the sequential and parallel program (without the schedule clause)
- Check they produce the same pictures
- Compare the timing for each case
- Try to use the schedule clause as described in the next slide
- Leave in the parallel program the schedule clause that provided the best results

Schedule clause

- Study the effect of using the schedule clause of the for directive:
  - Measure the time for the generation of each picture sometimes using the schedule(static,tt) clause and others the schedule(dynamic,tt) testing different values of tt
  - Compare the times for every picture with the ones in the parallel program without using the schedule clause, and with the ones of the sequential version
  - Include a **comment at the end of the code with the most relevant results obtained and a reasonable explanation of them**
    - Especify: processor model and number of cores

Handout

Upload to Campus Virtual the file
`mandelbrot_paralelo.c`
before two weeks
starting from today

And remember that **cheating is a bad idea!**

We'll check it.