Mandelbrot and K-means

Andrea Farruggia[*]

April 20, 2010

In this lesson we'll see two interesting topics, on their own and especially from the point of view of parallel programming: *Mandelbrot set* and *K-means*. After a general introduction, we'll focus on the features and issues that makes them as a good "sandbox" for discussing and implementing solutions to general and broad problems in parallel programming.

Mandelbrot Set

A Mandelbrot set $\mathscr{M}$ is a set of points in the complex plane which forms a *fractal*.

A "Fractal" is a structure which, among other properties, is *self-similar* to itself; in informal terms, a fractal is a structure (say, an image) that shows itself whatever "factor of magnifying" is used to look at it.

Many fractals have a simple, *recursive* definition, and Mandelbrot isn't an exception. We define a succession of complex values $z_0(c), z_1(c), \ldots$, such that

$$z_{n+1}(c) = z_n^2(c) + c$$

where $c \in \mathbb{C}$ is a parameter.

We say that $c \in \mathscr{M}$ if the modulus of the sequence $z_0(c), z_1(c), \ldots$ doesn't diverge to ∞; formally:

$$\mathscr{M} = \{c \mid \exists d \in \mathbb{R} : \forall n, |z_n(c)| \leq d\}$$

Well-known images like figure 1 are obtained interpreting the complex plane as a bitmap ($c = x + iy$) and plotting in black a pixel if the corresponding point c is in $\mathscr{M}$, otherwise plotting it with a color that reflects the "speed" with which it diverges (the number of iteration needed in order to detect divergence).

We outline some important properties of the Mandelbrot set:

- it can be shown that, if $c > 2$, then $c \notin \mathscr{M}$, so the Mandelbrot set is confined in a circle with radius 2; this is paramount to let $d = 2$ in previous definition.

[*]Somewhat edited by M.Coppola, April 20, 2010

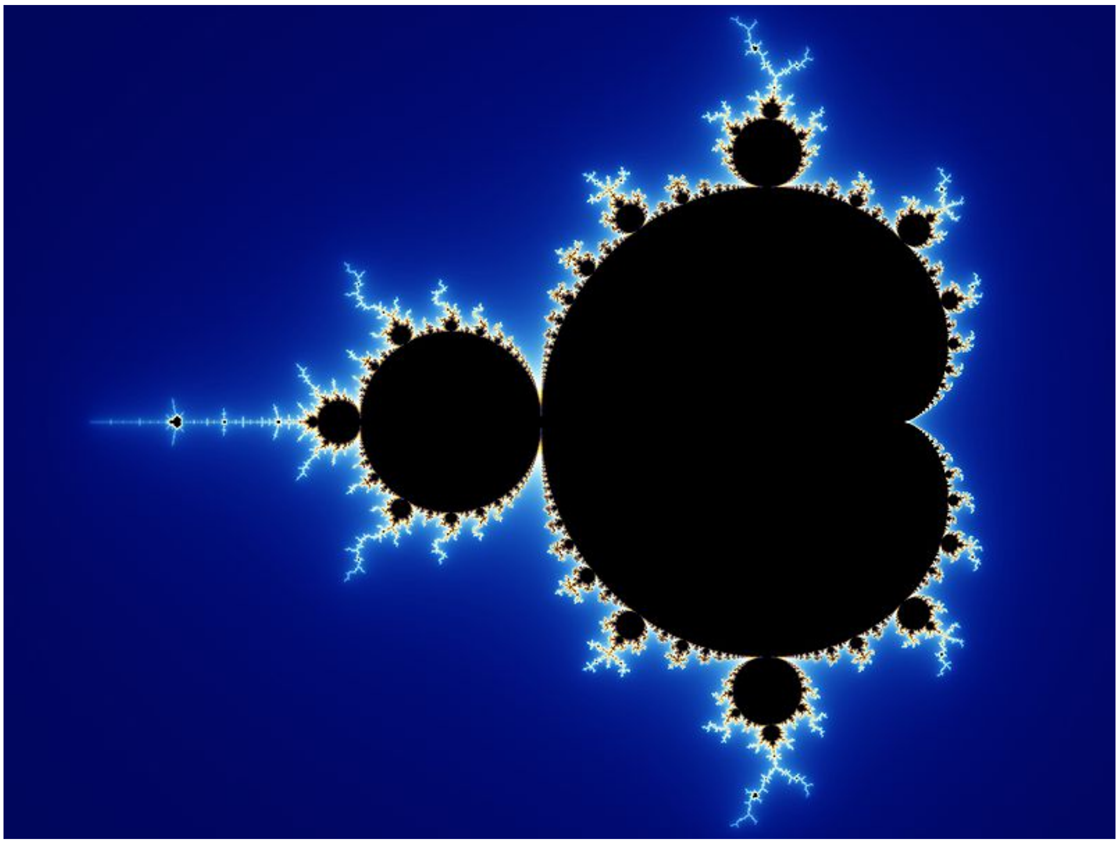

Figure 1: A plot of the Mandelbrot set

- the set it is roughly composed by a superimposition of three circles (or a circle and a cardiod figure), but it has smaller and smaller ramifications that start from the borders of the circles and connect to regions of the set which resemble the full Mandelbrot set in a smaller scale;
- the set is *connected*, which means that, for each couple of points $c_1, c_2 \in \mathscr{M}$, it exists a "path" entirely composed of points within $\mathscr{M}$ that connects c_1, c_2 ; the existence of these paths tells us that the seemingly separated copies of the set (which can be proved to be no artifacts) are actually linked to the main set by a stripe so thin that we cannot observe it at the resolution used to approximate the set.[1]
- it is not possible to explicitly compute all the points in $\mathscr{M}$ without using the recursive definition (e.g we can prove that some parts of the complex plane do/do not belong to the mandelbrot set; but we cannot prove it for arbitrary sets of points close to the set boundary).

[1]Addition: the proof that the set is connected also tells us that it has no holes in it.

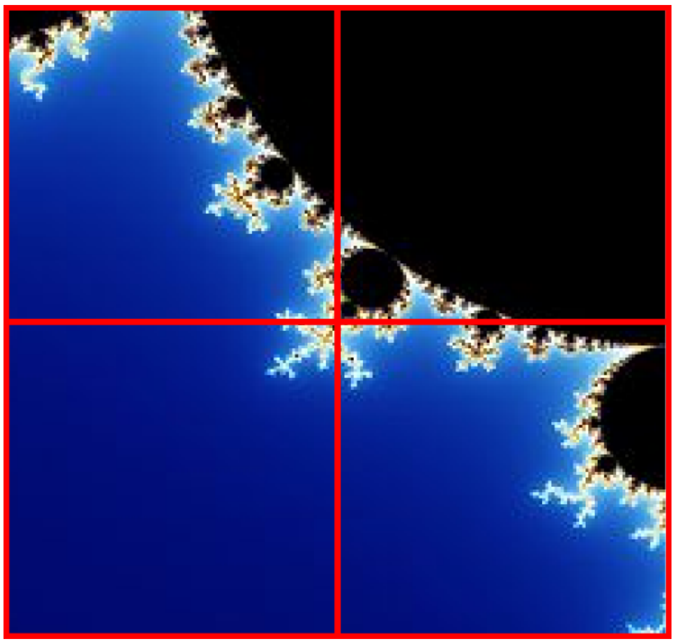

Figure 2: An example of unbalancing. The top-right region requires more iterations of the escape algorithm with respect to the others, especially the bottom-left one

Since we can't calculate explicitly $\mathscr{M}$, we have to *approximate* it. A common algorithm to use is the *escape time algorithm*. We compute iteratively the values $z_0, z_1, \ldots$, until which we have reached a critical 'escape' condition (and so the succession in that point diverges) or the number of values computed for that point is greater than a fixed threshold (and in that case it does not diverge).

The escape conditions $|z_n| > 2$ can be made simpler. Because no complex number with a real or imaginary part greater than 2 can have modulo less than 2 and be part of the set, a common condition is to escape when either coefficient exceeds 2, thus avoiding to explicitly compute the modulus of z_n. It is clear that the points that reqire the greater computational efforts are those in $\mathscr{M}$, followed by those which are very close to the set boundary.

Parallelization of the algorithm We can then use *data parallelism* to implement a parallel application that computes the colors of the points of a certain region of the complex plane. We can note several features of the problem that makes it worth considering for our purposes:

- we can tune the maximal number of iterations in order control how much the problem is computational intensive;
- the problem is heavily load-unbalanced, because the computation of some regions is much more demanding than in others (see the figure 2);
- because of the structure of the Mandelbrot set, it isn't possible, in general, to predict

which regions are more demanding than others;

In order to implement such parallel application, we have to answer to some questions:

- in how many regions (one for each task) do we have to subdivide the space?
- is the subdivision of the complex plan *static* or *dynamic*?

We make a few remarks

- As far as the first point is concerned, to allocate more tasks than processes gives us the opportunity to use a master-worker scheme in order to alleviate the unbalancing by letting the processes with "lighter" regions to fetch more tasks while others processes are stuck with "heavy" regions. Choosing the number of tasks to generate means setting the *granularity* of the computation.
- while submitting a task to a node it requires merely to send a few floating point numbers (the coordinates of the top-left and bottom-right corners, plus the expected resolution), sending back a result requires the communication of a *full* array of values (one per each pixel computed).

A more complicated scheme, which implements a dynamic generation of the tasks, use *divide and conquer*: when a task is assigned to a process, it is subdivided in K regions, and each process starts to compute one of them; if it is "heavy", then it is assumed that even the other K-1 subregions are heavy and then are submitted for execution to the emitter in order to not overload the current process.

Since "heavy" regions are mixed with "lighter" one in an unpredictable way, the assumption aforementioned can be wrong, thus leading to only a small gain. On the other hand, the offloading overhead is low, and in the case that there is at least another subregion as heavily loaded as the current one, the gain in load balancing is significant.

There needs to be some mechanism for detecting heavy load within a region computation (i.e. a detection mechanism like polling, interrupts, together with an evaluation function to decide when splitting the region is likely to result in a performance gain).

Resources

- the wikipedia page, with example pseudo code for the Mandelbrot function
  `http://en.wikipedia.org/wiki/Mandelbrot_set`
- the PBM ASCII-based graphic format, which can be easily generated via printf() to view mandelbrot sets computed by a program
  `http://en.wikipedia.org/wiki/Netpbm_format`