Problem set 1, Part 1

TDT4200, Fall 2016

Deadline: 2016–08–31, 20:00

Evaluation: Pass / Fail

Delivery: ItsLearning.
If you do not have access to It's Learning by Friday the 26. Aug, contact Prof. Anne C. Elster at elster@ntnu.no or the TA at tdt4200undass2016@gmail.com with TDT4200-NO-ITSLEARNING in the Subject line.
Deliver exactly two files via ItsLearning:

- *yourNTNUusername_ps1.pdf*, with answers to the theory questions.
- *yourNTNUusername_code_ps1.*{*zip* |*tar.gz* |*tar*} containing your solution to the programming tasks.

1 Theory

1. List and describe the six computer architectures classified in Flynn's taxonomy.
2. You want to parallelize the work done in the loop below. It turns out that the work done by `do_stuff()` is inherently serial for iterations i = 0–250, but you are able to fully parallelize the work done on iterations i = 251–999. Describe the possible speedup in terms of $\mathrm{T}_{\mathrm{parallel}}$, $\mathrm{T}_{\mathrm{serial}}$ and p, according to Amdahls law.

```
for(int i = 0, i < 1000, i++) {
  do_stuff()
}
```

3. What is the difference between parallel computing and distributed computing (according to Pacheco)
4. Non-graded, but useful for learning intro. concepts: Problems 1.3 and 1.4 in Pacheco

2 Programming, C Basics

In this part, your goal is to implement a program which determines whether or not a complex number diverges when computing fractals. A starting point has been provided in `complex.c`. Run `make run` to compile and run. C99 has support for complex number arithmetic. You are **not** allowed to use this. Remember that (-i) * (-i) = 1 (fundamental property of imaginary unit)

1. Create a `struct` for complex numbers called complex_t (use `typedef`), with double fields for the real and imaginary parts.
2. Implement the function `multiply_complex`. The function takes two complex numbers as arguments, and should return their product. Complex multiplication can be reduced to the following 4 real multiplications:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i$$

3. Implement the function `absolute_complex`. The function takes a complex number as an argument, and returns the absolute value. The absolute value of a complex number $z = a + bi$ is defined as:

$$|z| = \sqrt{a^2 + b^2}$$

4. Implement the function `create_random_complex_array`. The function takes a single integer `size` as an argument which determines the array size. The function should return an array of n (`size`) complex numbers. The range of the imaginary and real parts should be restricted to [-3, 3], which is the most interesting range for classic Mandelbrot sets.

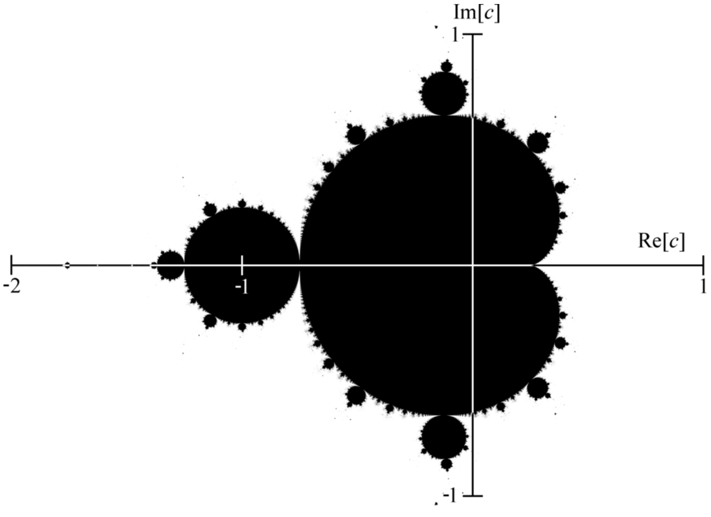

Figure 1: A mathematician's depiction of the Mandelbrot set M. A point c is colored black if it belongs to the set, and white if not. Re[c] and Im[c] denote the real and imaginary parts of c, respectively. Source: `https://en.wikipedia.org/wiki/Mandelbrot_set#/media/File:Mandelset_hires.png`

5. Implement the function `fractal_test_array`. The function takes one array of complex numbers, as well as the size of the array. The function should return a new array, where each element is either 1 or 0 depending on whether or not the corresponding element in the input array belongs to the Mandelbrot set (for the first iteration). That is,

$$\text{out[i]} = \begin{cases} 1 & \text{if } |in[i]| \geq 2 \\ 0 & \text{otherwise} \end{cases}$$

6. Test your implementation of `fractal_test_array` with an array of size n = 10^7. Measure the wall clock time of your program. It's sufficient to use the unix shell command `time` for this:

```
time ./executable
```

For more accurate timings, you need to modify the code with timers.

7. You can improve the performace of your program with a small trick: Test for $a^2 + b^2 \geq 4$ rather than $\sqrt{a^2 + b^2} \geq 2$. Repeat Task 2.6 with this new implementation.

On standard x86 laptop, you should be able to see some improvement.