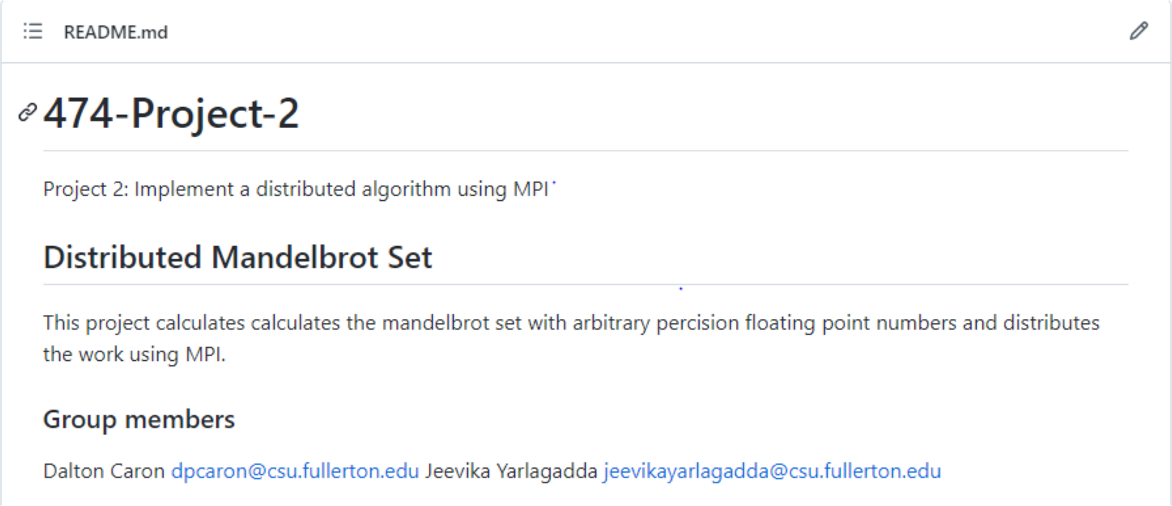
CPSC 474 - Project 2

GitHub url: https://github.com/CSUF-CPSC-Bein-FA21/project-2-cniles

Group Members: Dalton Caron, dpcaron@csu.fullerton.edu

Jeevika Yarlagadda, jeevikayarlagadda@csu.fullerton.edu

Submission for Project 2 - Distributed Mandelbrot Set



README.md

474-Project-2

Project 2: Implement a distributed algorithm using MPI

Distributed Mandelbrot Set

This project calculates calculates the mandelbrot set with arbitrary percision floating point numbers and distributes the work using MPI.

Group members

Dalton Caron dpcaron@csu.fullerton.edu Jeevika Yarlagadda jeevikayarlagadda@csu.fullerton.edu

Summary:

This project calculates the mandelbrot set with arbitrary precision floating point numbers and distributes the work using MPI.

Pseudo Codes:

The below are the two main algorithms used:

```
Algorithm 1: calculatePartition(c, offset, size, maxIterations)

Input  : a coordinate $c$, offset = $\frac{boundary}{size\ 1}$, $size$ of image partition
Output: an image $I$ of dimensions $size \times size$ and entropy $e$
$cx \leftarrow c.re$
for $x$ 0 to $size$ 1 do
```

```
    for $y \leftarrow 0$ to $size$ 1 do
        $z \leftarrow 0$
        for $iteration \leftarrow 1$ to $maxIterations$ do
            $z \leftarrow z^2 + c$
            if $z > |4$ then
                break
        if $iteration = maxIterations$ then
            $I[x][y] = black$
        else
            $I[x][y] = colorFunction(iteration)$
        $c.re = c.re + \text{offset}.re$
    $c.re = cx$
    $c.im = c.im - \text{offset}.im$
Compute the entropy of $I$ and store it as $e$
return $I, e$
```

Algorithm 2:

performIteration(*S*)

Input : a data structure of

settings *S*

Output: a computed image *I* and coordinate of partition with most entropy *c*

```
Populate and send the partitions to the follower processes
Receive each partition into the set of partitions $P$
$I \leftarrow (S.size, S.size)$
$maxEntropy \leftarrow$
foreach $p \in P$ do
    Combine partial solution $p.I$ into $I$
    if $p.entropy > maxEntropy$ then
        $maxEntropy \leftarrow p.entropy$
        $successorPartition \leftarrow p$
$c \leftarrow successorPartition.c$
```

```
return $I, c$
```

Compile the Code and Run:

- Code is written in C.
- Used makefile for easy compilation.

The user must have GCC, libgmp, and make installed to compile the program. The user's compiler must comply with POSIX standards. To compile the program, use the Makefile as shown below.

Command to compile: make build

This command produces the a.out executable file.

Command to run: make run

Depending on the parameters, the algorithm may execute in a few seconds or few hours. The default parameters in the Makefile run in a few seconds on modern machines. The Makefile default is executed using the above command.

To change the number of processors or the iterations, please go to the Makefile file in the directory and change the inputs as mentioned in the readme file.

Snapshots of the code with test data:

Test data 1:

Test data 2:

Test data 3: