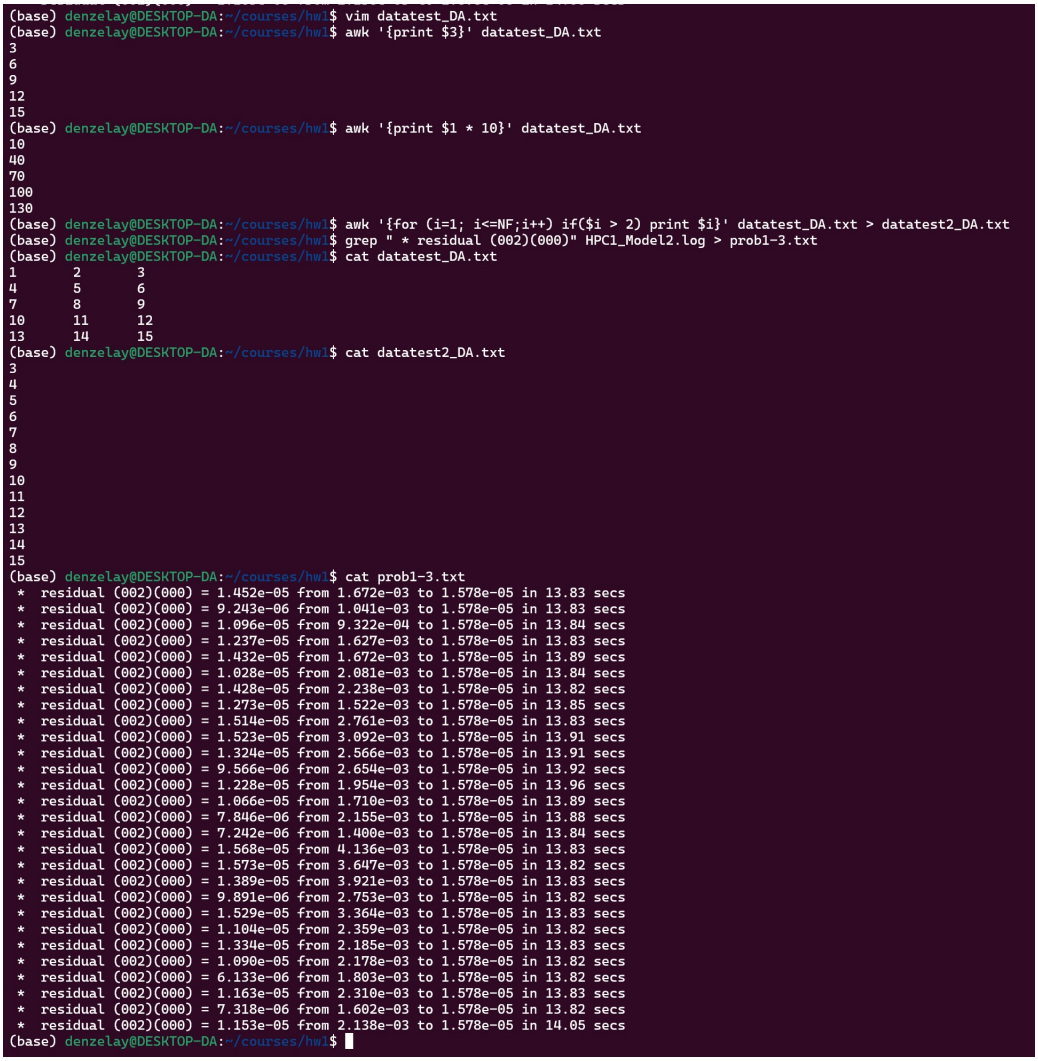
High Performance Computing 1
Assignment 1: Write Up
September 24, 2022

Name: *Denzel Ayala*

Problem 1

For the fourth part of this problem I obtained all the residues from the provided output file. For the cumulative run time I used a for loop in ***awk*** to add up all the provided times in the second to last column. In the case of count I gave two possible values. The first was the total number of rows extracted from the original file. The second count given was the addition of all the residues. Below is an image containing the commands I entered as well as the contents of the files requested. My code is slightly different from what is being displayed. For the screenshot I limited the residues gotten by grep using *…(002)*. In the submitted code every residue is printed to the file and the output count and runtime reflect that.



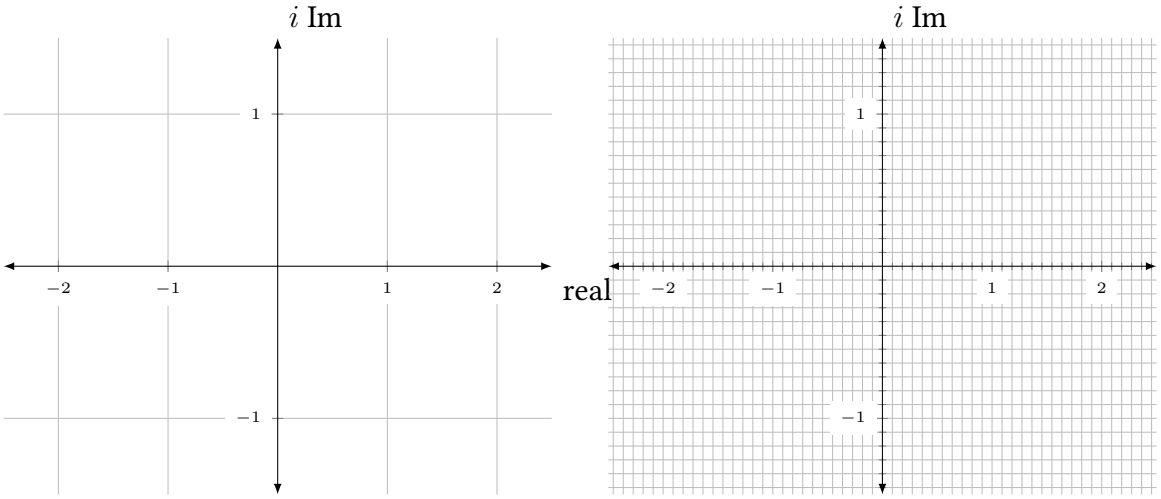
Problem 2

1. **Specify Problem Requirements**

   The goal is to plot the Mandelbrot set by calculating series convergence (eq 1) at every point in a space. For this problem we will be working with complex numbers. They consist of a real and *imaginary* component. These two components are orthogonal to each other. As a result the real and imaginary components can be treated as an $\hat{x}$ and $\hat{y}$ axis, respectively.

$$z_i = z_{i-1}^2 + c \tag{1}$$

   We are plotting within a range of $-2 \leq x \leq 2$ and $-1 \leq y \leq 1$. The plot range will be divided into individual points, that I will call ***pixels***, where convergence is tested. The ordered combination of the pixels form a mesh that determines the resolution power of the calculation. The plot on the left would be a representation of a low resolution mesh while on the left is a high resolution mesh.



   The convergence condition at a given pixel is $|z| \leq 2$. For understandability, an iteration will be a singular calculation of equation (1), where z_i is the output of each iteration. A round will consist of iteration till divergence or assumed convergence after 1k-10k iterations. For the first iteration of each round $z_{i-1} = z_0 = 0$. Meanwhile, c is given by the plot location of each pixel, or more tangibly each intersection (or square) in the plots above.

   The area is normally found by multiplying a length and a width. This would be difficult in the case of the Mandelbrot plot as its features are complex and very small. So a pixel count approach can be used to approximate the area of the curve.

2. **Analyze the Problem**

   The inputs to this system is the number of pixels in the real and imaginary directions. The output was initially a text file with a single binary (0 or 1) result for each pixel. In this initial output configuration a 0 indicated convergence while a 1 indicated the series diverted at that point c. This was modified later to be an unsigned integer where the value of the integer

indicates how many iterations it took to break the boundary condition. This modification was made to include color into the plots produced

The area input is the vector that holds all the pixel values. The output is a count of 0's in the vector divided by the total number of pixels.

3. **Design the Algorithm**

   There are 3 parts to the problem. First you need to calculate the Mandelbrot series for a given point. The second part is to loop through each point in the plot. The step size may change based on how much resolution is desired. The final part is to calculate the area of the plot that converged. This was broken up into 3 algorithms as listed below.

Algorithm 1 Mandelbrot Series Calculator

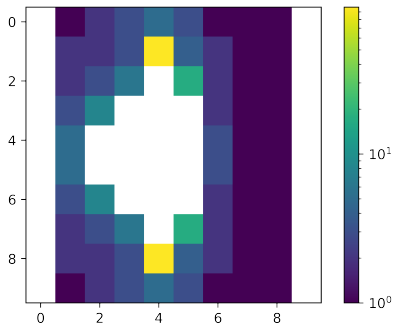
Require: $z_0 = 0$
  for $0 \leq i \leq 10,000$ **do**
    $z_{i+1} = z_i^2 + c$
    if $|z_{i+1}| > 2$ **then**
      Return i
    end if
  end for
  if $z \leq 2$ after loop **then**
    Return 0
  end if

Algorithm 2 Mesh Generator

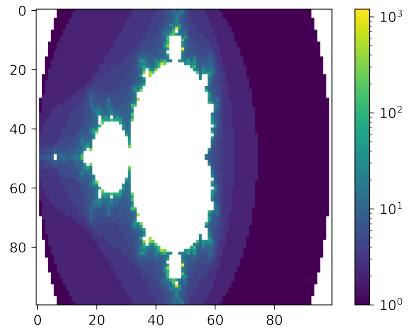
Require: mesh[ ] $\leftarrow$ Max Points ▷ a vector that holds all data points
Ensure: c is **ALWAYS** a point on the complex plane
  for $-1 \leq j \leq 1$ **do** ▷ This is counting through y
    for $-2 \leq i \leq 2$ **do** ▷ This is counting through x
      mesh[i] $\leftarrow MandelbrotSeries(c)$ ▷ input is c
      $c \leftarrow c + 1$
    end for
  end for

Algorithm 3 Area Finder

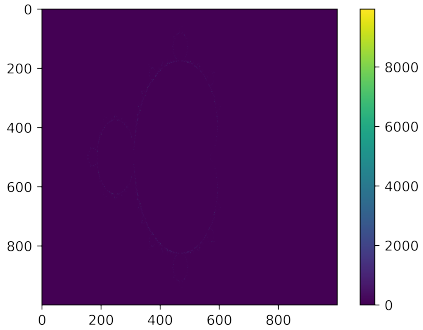
Require: totArea & mesh[size] are constant
  for i in mesh[ ] **do**
    if mesh[i] $= 0$ **then**
      Count $+= 1$
    end if
  end for
  Return $Area = totArea\left(\frac{Count}{mesh[size]}\right)$



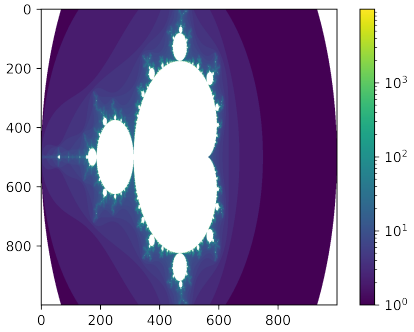
(a) 10x10 resolution



(b) 100x100 resolution



(c) 1000x1000 Linear

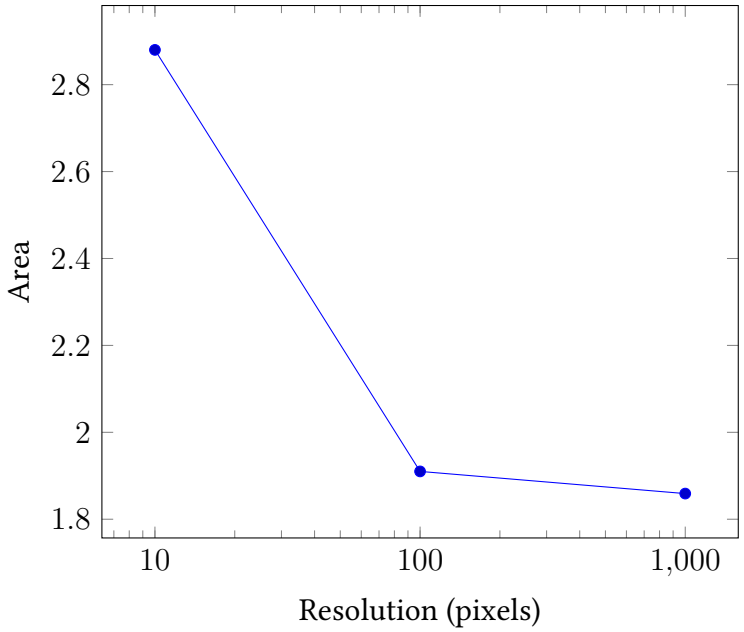


(d) 1000x1000 resolution

4. **Implement the Algorithm / Test and Verify the Completed Program**

   The algorithm compiled and ran. The output text files were put into python and turned into matrices that were plotted using Matplotlib in python. The resolutions were separated by an order of magnitude. The low resolution plot divided the x **and** y axis into 10 pixels. The middle resolution into 100 pixels, and the high resolution divided it into 1000 pixels.

   In these images the white corresponds to a convergent pixel. While at the low resolution there is some resemblance to the general shape of the MB series, as the grid resolution increases more details of the fractal come into focus. As soon as the mid resolution the more detailed fine structures of the MB set can be seen. At the high resolution the image has take its iconic look. The images were plotted on a logarithmic color scale. There are several points along the "edge" of the fractal pattern that take several thousand iterations to converge. As a result the linear color scale plot is flat and uninformative. This is becasue the majority of points converge within a couple hundred iterations. The area calculations showed that as the resolution increased the area decreased and converged towards 1.86 units.



Problem 3

$$\text{let } avg = \frac{\text{average cycles}}{\text{word access}} = f_c \times \frac{\text{cache cycles}}{\text{word access}} + (1 - f_c)\frac{\text{main memory cycles}}{\text{word access}}$$

$$\text{FLOPS} = \frac{2.3\text{GHz } \rho_{wm}}{avg}$$

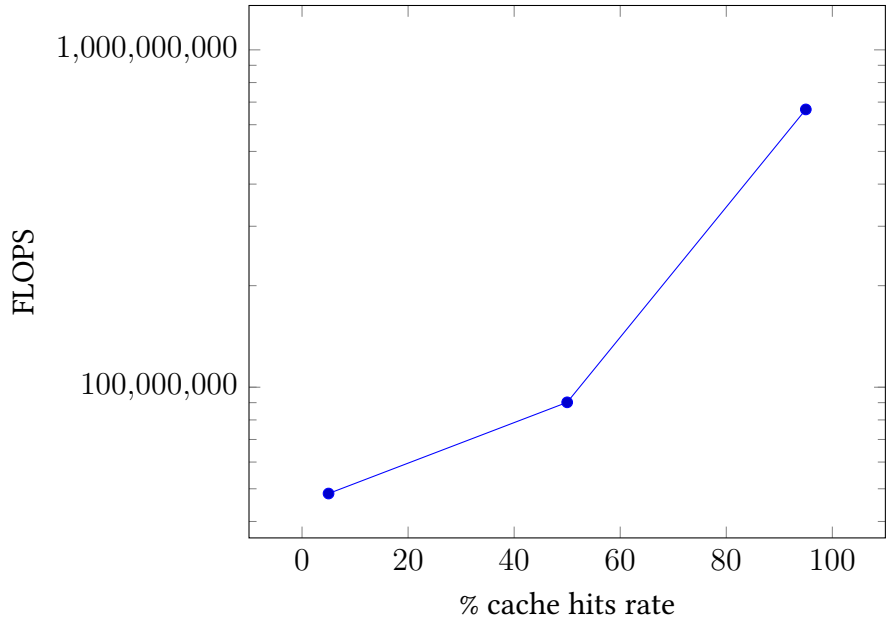
$$\text{units: } \frac{\cancel{cycles}}{s}\frac{flop}{\cancel{wordaccess}}\frac{\cancel{wordaccess}}{\cancel{cycles}}$$

Rearranging all the equations and plugging in given constants we get a final equation as seen below.

$$\text{FLOPS} = \frac{2 \times 2.3\text{E}{-}10}{2f_c + 100(1 - f_c)}\frac{flop}{s}$$

The answers will be given without showing the plugin step.

1. 6.66E8 $\frac{flop}{s}$
2. 9.02E7 $\frac{flop}{s}$
3. 4.84E7 $\frac{flop}{s}$
4. Plot given below.



5. The system with 95% cache hit rate has the best performance. It is able to complete an order of magnitude more floating point operations per second than the 50% cache hit rate system.

Problem 4

1. L1 cache $<$ L2 cache $<$ main memory

2. main memory $<$ L2 cache $\leq$ L1 cache

3. Let us think of data as water that needs to be transported. If we work with the assumption that all water being transported is moving at a **constant** rate we can look at latency and bandwidth as properties of an empty pipe that transports the water. Latency is the length of the pipe from your starting point to your end point. The longer the pipe is the longer the delay till water starts flowing out the end point. So low latency means your data gets moved faster. On the other hand bandwidth is akin to the width of the pipe. A larger pipe can transport **more** water over the same length of pipe. This makes it so that even if you have a large latency you can still transfer a lot of data/water.
   Low latency and high bandwidth would give the fastest performance. In practice this can be difficult to achieve.