2-Dimensional Heat Distribution - 100 points

For this assignment, you are required to model how heat spreads through a laminar material on 2 dimensions. You need to model the material using a 2 dimensional grid using a **`numpy`** array and calculate how the heat flows through it. We are going to use an iterative method that determines the value of the current cell using the 4 cardinal neighbors of the cell.

1. Heat is generated at the left wall of the material, so the left wall is at a constant temperature. Every other cell starts off at 0.

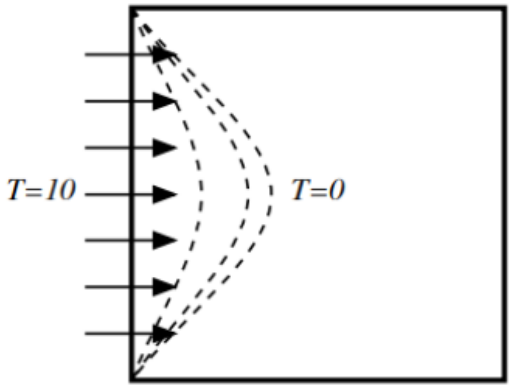


Figure 1: Temperature at the left wall

2. At every cell, the current temperature is calculated using the temperature at the cardinal neighbors at the previous state, using the following equation:

$$temp[i,j] = \frac{1}{4}(oldTemp[i-1,j] + oldTemp[i+1,j] + oldTemp[i,j-1] + oldTemp[i,j+1])$$

3. This calculation is repeated several times, for each cell.
4. The calculation ends at convergence, where two consecutive iterations result in the same values.

Specifications

For this assignment, you will write a program that involves the third party Python libraries numpy and matplotlib. You will turn in 2 programs - the first, called **`heatDiff.py`** will estimate 2D heat distribution with serial code, and the second, called **`parallelHeatDiff`** will parallelize your serial code.

- **heatDiff.py**

  – Get the starting temperature from the user. This will be an integer. (5 points)

– Create a 2-dimensional **`numpy`** array. The grid will have as many rows and columns as the starting temperature. You will also need *halo cells*. Halo cells - one extra row each on the top and the bottom and one extra column each at the left and the right. (5 points)
– Set the leftmost column (halo column) values to the starting temp. Set all the other values in the grid to 0. (5 points)
– You would need 2 of these grids, one for the previous state and one for the current state. (5 points)
– Use the given equation to calculate the temperature at each cell of the current state grid using values from the previous state grid. (10 points)

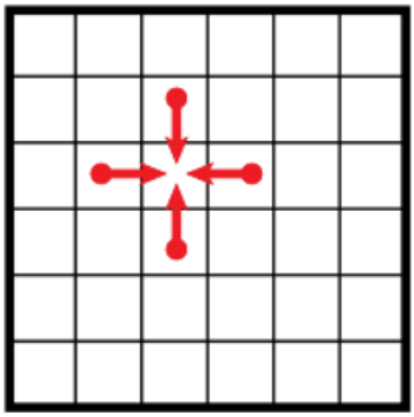


Figure 2: Calculating temperature at a cell

– While calculating the temperatures at each iteration, do not change the values of the halo cells. These cells will stay at starting temperature for the left wall and 0 otherwise.
– At the end of the iteration, check if the current state is EXACTLY the same as the new state. If so, we're done. Otherwise repeat the process. (5 points)
– Sometimes, we will not get convergence for a very long time. In the interest of time, we will stop at 3000 iterations if we don't see convergence by then. (5 points)
– Time your code to figure out how long the program takes to run. (5 points)
– Once the grid has been finalized, plot each point onto a graph.

- ∗The color of the point is determined by the temperature of the cell at the end of the last iteration. (5 points)
- ∗The plotting should be done with a single colormap and a single call to the scatterplot function. Look at the docs for **`plt.scatter`**. This will greatly reduce the runtime. (15 points).
- ∗Do not use **`imshow`**, this has anti-aliasing that will give you a different smoothed output, which is not what we want. imshow is also a crutch.

– We have used 8 colors for our graphs where **darkred** is the hottest spot and **darkblue** is the coolest spots. Divide the temperatures equally on the scale between 0 and the starting temperature.

- **parallelHeatDiff.py**

  – Go through Steps 1-4 exactly the same way.
  – Create a Pool of processes. (5 points)
  – Write a function that takes in a smaller subset (a mini grid) of cells and calculates the heat in the mini grid for the next iteration, for one step. For this step, you may partition the grid however you please. But, when you pass the partition in as a parameter, make sure you include halo cells all around it. (5 points)
  – Use either the asynchronous apply or map function to use the above function to apply the iterative method in parallel. (20 points)
  – Note that every time, you have to communicate the results of that process to the overall grid.
  – Time your program to figure out how long it takes to run. (5 points)
  – Note: Depending on your machine power or the power of your VM, and your method, this might take longer than the serial method, even though in theory, it should not.

Sample Output

This section shows the sample output for the program for 2 different runs, one when the temperature is 30 and once when it is 150. The output looks more and more like a heatmap when the initial temperature values are higher. For reference, the colors used for the heatmap are - **darkred**, **red**, **orange**, **yellow**, **lawngreen**, **aqua**, **blue**, **darkblue**

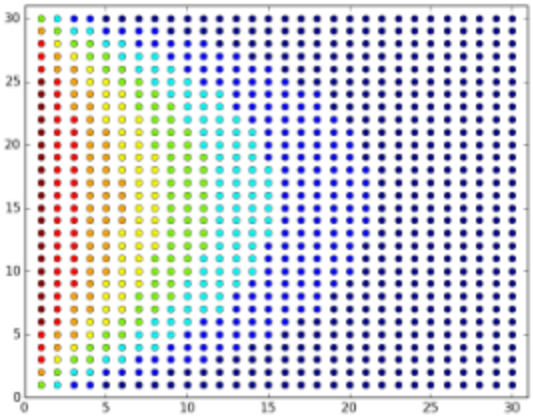
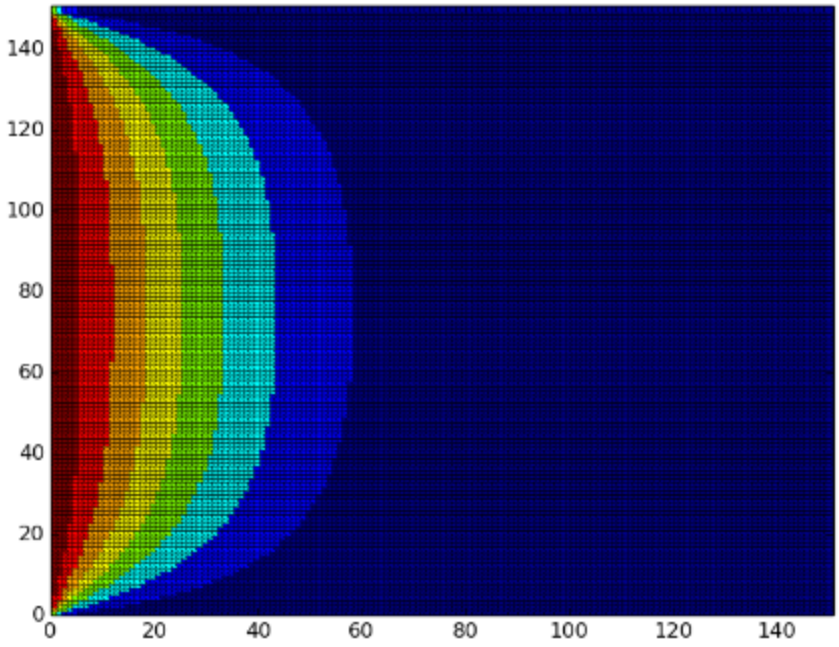


Figure 3: Starting Temperature: 30



General Guidelines

•Call your files **heatDiff.py** and **parallelHeatDiff.py**. These are the only files you will be delivering.
•If we have listed a specification and allocated points for it, you will lose points if that particular item is missing from your code, even if it is trivial.
•Your program should load and run without issues. Every interpretation error will result in a loss of 5 points each.
•You are restricted to standard Python (built-ins), **`numpy`** and **`matplotlib`**. Use of any other libraries would result in loss of 10 points per library.