HOMEWORK – 4

Name:Shushhma Dewie Koppireddy

ID:02190512

PROBLEM – 1

Redo the integral example for the double integral case on a rectangular domain.

```python
import numpy as np

def f(x, y):
    return x**2 + y**2  # Example function

def midpoint_double_integral(x_start, x_end, y_start, y_end, nx, ny):
    hx = (x_end - x_start) / nx  # Step size in x direction
    hy = (y_end - y_start) / ny  # Step size in y direction

    integral = 0
    for i in range(nx):
        for j in range(ny):
            x_mid = x_start + (i + 0.5) * hx
            y_mid = y_start + (j + 0.5) * hy
            integral += f(x_mid, y_mid)

    return integral * hx * hy

a, b = 0, 2
c, d = 0, 2

n_workers = 4

nx_total = 100
ny_total = 100

x_splits = np.linspace(a, b, n_workers + 1)

worker_results = []

for i in range(n_workers):
    x_start = x_splits[i]
    x_end = x_splits[i + 1]

    subdomain_integral = midpoint_double_integral(x_start, x_end, c, d, nx_total // n_workers, ny_total)
    worker_results.append(subdomain_integral)
    print(f'Worker {i+1} integral value: {subdomain_integral}')

total_integral = sum(worker_results)

print(f'Total integral value: {total_integral}')

```

Output:

Worker 1 integral value: 1.4166

Worker 2 integral value: 1.91659

Worker 3 integral value: 2.91660

Worker 4 integral value: 4.41659

Total integral value: 10.666

PROBLEM – 3

Modify the mandelbrot spmd program such that the grid domain 1000 x 1000 is partitioned into chunks of *square* blocks.

```python
import numpy as np
import matplotlib.pyplot as plt
import time

max_iterations = 1000
grid_size = 1000
xlim = [-0.748766713922161, -0.748766707771757]
ylim = [0.123640844894862, 0.123640851045266]

start_time = time.time()
x = np.linspace(xlim[0], xlim[1], grid_size)
y = np.linspace(ylim[0], ylim[1], grid_size)
x_grid, y_grid = np.meshgrid(x, y)
z0 = x_grid + 1j * y_grid
count = np.zeros_like(z0, dtype=float)

z = z0
for n in range(max_iterations):
    mask = np.abs(z) <= 2  # Create a mask for points that are still in the iteration
    count += mask  # Update count only for points that are still valid
    z[mask] = z[mask] * z[mask] + z0[mask]  # Update only valid points

count = np.log(count + 1)  # Avoid log(0)

cpu_time = time.time() - start_time
plt.figure(figsize=(6, 6))
plt.imshow(count, extent=(xlim[0], xlim[1], ylim[0], ylim[1]), cmap='jet')
plt.axis('equal')
plt.axis('off')
plt.colorbar()
plt.title(f'{cpu_time:.2f} secs (serial)')
plt.show()
```

Output:

We can see a square plot showing colorful patterns will vary depending on the escape rates of the points , and the image

will be in green.The title will include the execution time in seconds, indicating how long the calculation took

The execution time: 28.92 seconds (serial).

This the link for all the code Here