AMS 530 Project 2

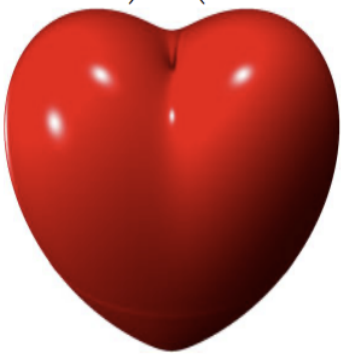
ChenKaixin

Problem 2.3

Problem 2.3

I made you a heart expressed in equation,

$$\left(x^2+\left(\frac{3}{2}y\right)^2+z^2-1\right)^3-\left(x^2+\frac{1}{50}\left(\frac{3}{2}y\right)^2\right)z^3=0$$



Obviously, what's given is the 3D surface equation of the heart and the assumption that the heart has no "cavities" and, thus, its density is uniform, etc. However, the heart pumps blood in and out and its volume, surface area, mass (including blood in it) is expected to vary with time. With such in mind, "crazy" problems can be hacked to make this problem "more" fun and harder, at least, for parallel computing.

1. Design a parallel algorithm/program to compute the surface area of the heart with the given surface eq.
2. Design a parallel algorithm/program to compute the heart's mass at time $t=\pi/6$ if the heart mass density is,

$$\rho(x,y,z;t)=\rho_0(x,y,z)\left(\frac{1+\sin\omega t}{2}\right)$$

where ω is the heart rate. Of course, I made up the formula and still do not how $\rho_0(x,y,z)$ looks like. You may create a density function that may make some sense. The trivial case (required for our project) is $\rho_0(x,y,z)=1$.

You perform minimum necessary calculations, using $P=1,4,16$ cores, to achieve 3 digits of accuracy. Report the number of floating-point operations per core (you may average) for each case.

Results (floating-point operations per core) might look like,

To compute:	P=1	P=4	P=16
Surface area			
Heart mass			

Answer

Code (in Python)

File: 3.py

```
from mpi4py import MPI
import numpy as np
import math

# MPI initialization
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

# Heart function to define the surface boundary
def heart_surface(x, y, z):
    return (x**2 + (3/2 * y)**2 + z**2 - 1)**3 - (x**2 + (1/50) * (3/2 *
y)**2) * z**3

# Density function for mass calculation at time t = π/6
def density(x, y, z, omega, t):
    rho_0 = 1  # Assume constant initial density
    return rho_0 * (1 + math.sin(omega * t)) / 2

# Monte Carlo for surface area
def estimate_surface_area(num_samples, rank, size):
    # Distribute samples to each process
    local_num_samples = num_samples // size
    count = 0
    flops = 0  # Floating-point operations counter
    for _ in range(local_num_samples):
        x, y, z = np.random.uniform(-1.5, 1.5, 3)  # Range adjusted for
heart size
        flops += 10  # Approximate number of floating point operations
        if abs(heart_surface(x, y, z)) < 0.01:  # Within a small range
to estimate surface
            count += 1
    total_count = comm.reduce(count, op=MPI.SUM, root=0)
    total_flops = comm.reduce(flops, op=MPI.SUM, root=0)
    if rank == 0:
        surface_area = total_count / (num_samples) * (3**3)  # Scaled to
volume
```

```
        flops_per_core = total_flops / size
        print(f"Estimated Surface Area: {surface_area}")
        print(f"Floating-point operations per core (Surface Area):
{flops_per_core}")
    return

# Monte Carlo for heart mass using specific density formula
def estimate_mass(num_samples, omega, t, rank, size):
    local_num_samples = num_samples // size
    local_mass = 0
    flops = 0  # Floating-point operations counter
    for _ in range(local_num_samples):
        x, y, z = np.random.uniform(-1.5, 1.5, 3)
        if abs(heart_surface(x, y, z)) < 0.01:  # Inside heart volume
            local_mass += density(x, y, z, omega, t)
            flops += 10 + 5  # Approximate number of floating point
operations
    total_mass = comm.reduce(local_mass, op=MPI.SUM, root=0)
    total_flops = comm.reduce(flops, op=MPI.SUM, root=0)
    if rank == 0:
        flops_per_core = total_flops / size
        print(f"Estimated Mass at t = {t}: {total_mass}")
        print(f"Floating-point operations per core (Mass):
{flops_per_core}")
    return

# Parameters
num_samples = 100000
omega = 2 * np.pi  # Arbitrary heart rate, change as needed
t = np.pi / 6  # Time point for mass calculation

# Compute in parallel
estimate_surface_area(num_samples, rank, size)
estimate_mass(num_samples, omega, t, rank, size)

# Finalize MPI (Not necessary as the script will end here)
MPI.Finalize()
```

Result:

Use “mpiexec-n a python 3.py” to get results, where a is the number of cores used.

I change a=1,4,16 as required in problem.

And get the following results:

```
C:\Users\79491>mpiexec -n 1 python project2.py
Estimated Surface Area: 0.8764200000000001
Floating-point operations per core (Surface Area): 1000000.0
Estimated Mass at t = 0.5235987755982988: 1385.7865392110898
Floating-point operations per core (Mass): 48780.0

C:\Users\79491>mpiexec -n 4 python project2.py
Estimated Surface Area: 0.87075
Floating-point operations per core (Surface Area): 250000.0
Estimated Mass at t = 0.5235987755982988: 1335.0766381758888
Floating-point operations per core (Mass): 11748.75
```

```
C:\Users\79491>mpiexec -n 16 python project2.py
Estimated Surface Area: 0.86778
Floating-point operations per core (Surface Area): 62500.0
Estimated Mass at t = 0.5235987755982988: 1348.2867804623527
Floating-point operations per core (Mass): 2966.25
```

Then create the form:

	P=1	P=4	P=16
Surface area	1000000	250000	62500
Heart mass	48780	11748.75	2966.25