High Performance Computing (HPC)
Homework Assignment

Instructor: Dr. Bhaskar Chaudhury
Teaching Assistant: Libin Varghese, Ayushi Sharma, Vansh Joshi

March 25, 2025

General Instructions

- This assignment focuses on parallel programming concepts, optimization, and scalability analysis.
- Please provide your solutions by uploading your report's PDF and relevant code files.
- Make sure all plots, tables, and commentary are neatly formatted.
- Bonus marks will be given based on the performance of the parallel code.
- You will get two weeks for solving, testing and improving the code and making the report.

---

1 Problem Description

Given a set of scattered points in a 2D domain with known function values:

$$S = \{(x_i, y_i, f_i) \mid i = 1, 2, \ldots, N\}$$

Where (x_i, y_i) are spatial coordinates and f_i are known function values, the task is to interpolate these values onto a structured mesh grid of size $M \times M$. For simplicity, we will assume $f_i = 1$.

The structured mesh consists of regularly spaced grid points:

$$G = \{(X_i, Y_j) \mid X_i = i\Delta x, \quad Y_j = j\Delta y, \quad i, j = 0, 1, \ldots, M-1\}$$

where Δx and Δy define the uniform grid spacing and $0 <= x_i <= X_{max}$ and $0 <= y_i <= Y_{max}$. Here $\Delta x = X_{max}/M$ and $\Delta y = Y_{max}/M$. See Fig.1 for reference.

Typical use of this scheme is mentioned here.

1. **Computer Graphics and 3D Modeling**:
   - Used to generate smooth surfaces from scattered 3D points, such as those obtained from 3D scanning (e.g., LIDAR or photogrammetry). For example, turning a point cloud of a scanned object into a mesh for rendering or animation.

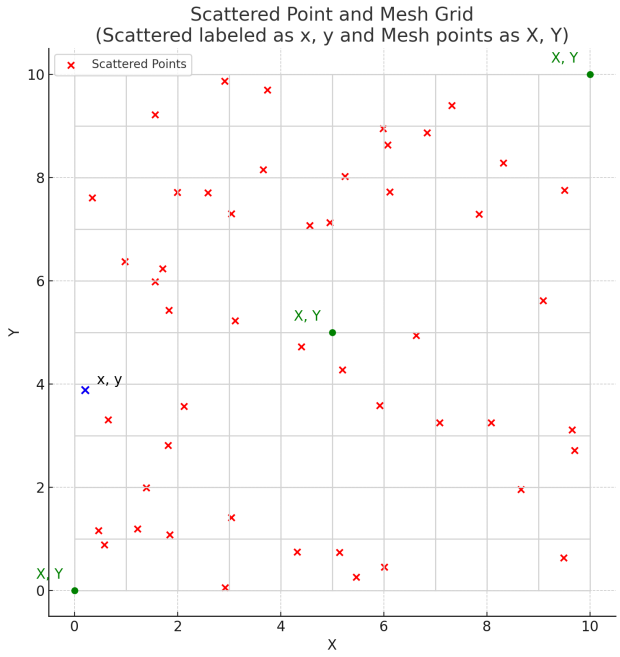

Figure 1: Figure of the domain with scattered points

2. **Scientific Visualization**:
   - In fields like meteorology or oceanography, scattered data points (e.g., temperature or pressure readings from sensors) are interpolated onto a mesh to visualize continuous fields over a region.
3. **Finite Element Analysis (FEA)**:
   - Engineers use this in simulations (e.g., structural analysis, fluid dynamics) to map scattered data (like material properties or experimental measurements) onto a structured mesh for numerical solving.
4. **Medical Imaging**:
   - Reconstructing surfaces or volumes from scattered data points, such as creating a 3D model of an organ from MRI or CT scan points.
5. **Machine Learning and Data Science**:
   - When dealing with sparse or unevenly sampled data, this scheme can help create a continuous representation for tasks like regression, visualization, or simulation.

2 Linear Interpolation

Linear interpolation is a simple **bilinear interpolation** method that distributes each scattered point's value to the **four nearest grid points** based on its relative position in the grid cell.

1. Identify the four nearest grid points (X_i, Y_j) surrounding a scattered point (x_i, y_i). 2. Compute the weights based on the relative distances of (x_i, y_i) to these grid points. 3. Distribute the value f_i proportionally to these grid points.

For a scattered point (x_i, y_i) located in a grid cell with corners at: - (X_i, Y_j) - (X_{i+1}, Y_j) - (X_i, Y_{j+1}) - (X_{i+1}, Y_{j+1})

The weight factors are computed as:

$$w_{i,j} = (1 - d_x)(1 - d_y)$$

$$w_{i+1,j} = d_x(1 - d_y)$$

$$w_{i,j+1} = (1 - d_x)d_y$$

$$w_{i+1,j+1} = d_x d_y$$

where:

$$d_x = \frac{x_i - X_i}{\Delta x}, \quad d_y = \frac{y_i - Y_j}{\Delta y}$$

Since the values of the grid point are not necessary to be stored, it should be calculated on the fly:

$$X_i = ((int)\frac{x_i}{\Delta x}) * \Delta x, \quad Y_j = ((int)\frac{y_i}{\Delta y}) * \Delta y$$

The interpolated function values at the structured grid points are accumulated as:

$$F(X_i, Y_j) += w_{i,j} f_i$$

$$F(X_{i+1}, Y_j) += w_{i+1,k} f_i$$

$$F(X_i, Y_{j+1}) += w_{i,j+1} f_i$$

$$F(X_{i+1}, Y_{j+1}) += w_{i+1,j+1} f_i$$

For Example, the value to be stored on grid point i,j depends on the value of weight wij, which is calculated using the area of the blue region as shown in Fig. 2.

In a serial implementation, interpolation is straightforward and requires two data structures: one for storing scattered points and another for storing interpolated values on a structured mesh. However, in a parallel implementation, special care must be taken to eliminate dependencies. This is because multiple scattered points may fall within the same or adjacent cells, leading to shared grid points and potential race conditions. To improve performance, these dependencies must be carefully managed. **This can be achieved by modifying the data structures or optimizing the parallel implementation to ensure efficient computation. Your final scores will be evaluated based on the leaderboard, and your results will be shared in Google Classroom. Your end goal is to increase the performance of the existing code for a large number of threads.**

3 Implementation Steps

1. **Read input data**: Load scattered points (x_i, y_i, f_i) from the file *input.bin*
   - The data in this file is in binary format to save space.
   - The data file given to you is for a domain of size 1 unit x 1 unit. All particles and grid points should lie within this domain.

2. **Initialize structured mesh**: Define the grid G and set all function values $F(X_i, Y_j)$ to zero.

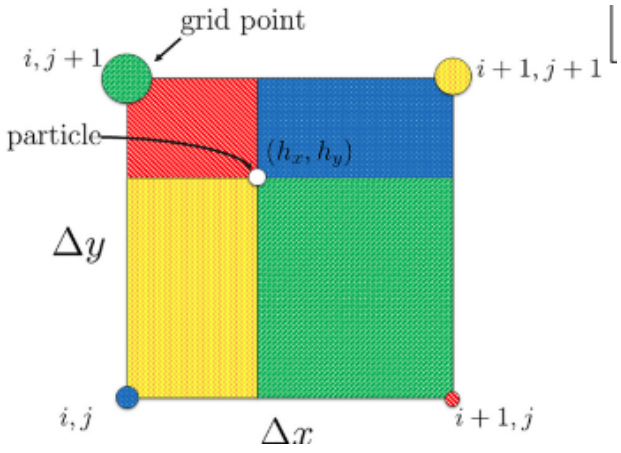

Figure 2: linear interpolation. Here h_x,h_y is equivalent to x_i,y_i

3. **Perform interpolation**:
   - Loop through each scattered point.
   - Identify the four nearest structured grid points.
   - Compute the weights w.
   - Distribute the function value accordingly.
   -Another way to do this is to Loop through each grid point.
   - Identify all the scattered points associated with the grid point.
   - Compute the weights w.
   - Distribute the function value accordingly.
   -This step can be performed in multiple ways. Think out of the box.
   -First, write a serial code and check for accuracy using the provided results.
   -Then, think of the dependency you will encounter when parallelizing the code.
   -**You are free to use any data structure or methods which will aid in parallelizing**.
   -Use your knowledge of HPC and remember the concepts of data locality. This will help in writing efficient and scalable parallel code.
   -Make sure that the input and output of the interpolation function are not altered.
   -This step will require careful planning and consideration of the bottlenecks.

4. ** Step 3** will be performed 10 times each time a new set of scattered points will be loaded from the input file.

5. **Output the interpolated grid**: Write the structured mesh values to an output file.

6. To check the accuracy of your code, use the input file provided with the codes and match the values with the output file. Always make sure your code is giving accurate results. **Speedup is the goal but not at the cost of accuracy**.

7. You have been provided with the code *input_fileMaker.c* that makes the input data file *input.bin*.
   -Use this code to create the input files with these configurations and perform the interpolation step.

   (a) Nx=250, Ny=100, points=0.9 million, Maxiter=10.

   (b) Nx=250, Ny=100, points=5 million, Maxiter=10.

(c) Nx=500, Ny=200, points=3.6 million, Maxiter=10.

(d) Nx=500, Ny=200, points=20 million, Maxiter=10.

(e) Nx=1000, Ny=400, points=14 million, Maxiter=10.

You can also edit the PIC_interpolation.c so that you don't need to load the file and instead directly generate the required points while debugging. But be careful to revert while submitting, as I need the code to accept the input file.

4 Analysis and Questions

1. Theoretical analysis

   (a) Compute the code balance

   (b) Write the pseudocode of the proposed implementation in the report along with an illustrative diagram of your implementation.

   (c) Perform complexity analysis and theoretical cache access analysis

   (d) Calculate the parallel efficiency.

   (e) If given **unlimited HPC resources**, how would you further optimize the implementation?

2. Experimental Observations

   (a) Compare execution time between the serial code provided and your implementation (No graph needed).

   (b) Plot graph of speedup vs cores/threads and execution time vs cores/threads for each of the five inputs. **To measure speedup, use the serial code of your implementation.** - Use as many threads as physical cores are available on the HPC machine. If you have access to machines with more threads, you can also test your code on that and generate plots. Ensure no one else is using the machine when running codes to get good performance. I recommend that you do not wait until the final submission date to test the code (2 graphs needed).

   (c) Plot execution time vs core graphs for the 5 problems to compare your implementation against the code shared (5 graphs needed).

   (d) Measure cache misses using some profilers such as vtune/valgrind/callgrind.

   (e) Run codes on HPC and lab machine (use only 4 threads). No graph is needed.

   (f) Analyze the impact of hyperthreading on performance. - Compare execution time when using only physical cores vs. using all logical cores.

   (g) Effect of different thread scheduling mechanism

3. Interpretation of results

   (a) Try to explain the speedup achieved and all the observations of the previous questions using your current HPC knowledge.

5 Submission Guidelines

- Submit a **PDF report** including:
  - Explanation of your approach.
  - Performance analysis.
  - Answer to analysis questions.
- Submit your **source code** in a compressed file (‘zip‘ or ‘tar.gz‘).
- The deadline for submission is **April 13, 2025 at 11:59 PM**.

6 Grading Criteria

- **Correctness** (30%) - The implementation correctly interpolates values. The accuracy of your codes will tested against a dataset not shared with you.
- **Novelty** (70%) - Since the basic structure of the code is already provided, you are required to come up with different data structures to store and access data or different parallelisation methods to increase the performance of the given code.

7 Remarks

With so much information and resources at your disposal, I expect sincere efforts. **All the best**.