CS 301
High-Performance Computing

Lab 03 – Serial Interpolation

Akshat Bhatt Gandhi(202201460)
Vivek Parmar(202201475)

February 24, 2026

Contents

1 Introduction

In this lab, we implemented a serial interpolation algorithm that maps scattered data points onto a structured grid. The goal was to understand how irregular data can be converted into a continuous field representation and to analyze the performance of the algorithm on both the Lab PC and the HPC cluster.

The interpolation method distributes the value of each scattered point to nearby grid nodes, producing a smooth representation. This technique is widely used in computer graphics, scientific visualization, simulations, and data analysis.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU: 12-core 12th Gen Intel(R) Core(TM) i5-12500
- CPU Max Frequency: 4600 MHz
- L1d Cache: 288 KB
- L1i Cache: 192 KB
- L2 Cache: 7.5 MB
- L3 Cache: 18 MB

2.2 Hardware Details for HPC Cluster (Node gics2)

- Architecture: x86_64
- CPU Mode: 32-bit, 64-bit
- Byte Order: Little Endian
- Total CPUs: 16
- Threads per Core: 1
- Cores per Socket: 8
- Sockets: 2
- NUMA Nodes: 2
- Processor: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
- CPU MHz: 1229.414
- Virtualization: VT-x

- L1d Cache: 32 KB
- L1i Cache: 32 KB
- L2 Cache: 256 KB
- L3 Cache: 20 MB
- NUMA node0 CPUs: 0–7
- NUMA node1 CPUs: 8–15

3 Problem Description

We are given a set of scattered points:

$$S = \{(x_i, y_i, f_i)\}$$

where (x_i, y_i) represent coordinates and f_i represents function values. For this assignment, all function values are:

$$f_i = 1$$

All points lie inside a 1×1 domain.

The objective is to interpolate these values onto a structured grid:

$$G = (X_i, Y_j)$$

with grid spacing:

$$\Delta x = \frac{1}{N_x}, \quad \Delta y = \frac{1}{N_y}$$

Each scattered point contributes to the four nearest grid nodes.

4 Bilinear Interpolation Method

Each scattered point lies inside a grid cell formed by four grid nodes. Instead of assigning the point value to a single node, bilinear interpolation distributes the value to the four surrounding nodes based on the point's relative position.

Let the distances of the point from the lower-left corner of the cell be:

$$l_x = x_i - X_i, \quad l_y = y_i - Y_j$$

The interpolation weights are computed as:

$$w_{i,j} = (\Delta x - l_x)(\Delta y - l_y)$$

$$w_{i+1,j} = l_x(\Delta y - l_y)$$

$$w_{i,j+1} = (\Delta x - l_x) l_y$$

$$w_{i+1,j+1} = l_x l_y$$

These weights ensure smooth distribution and preserve continuity across the grid.

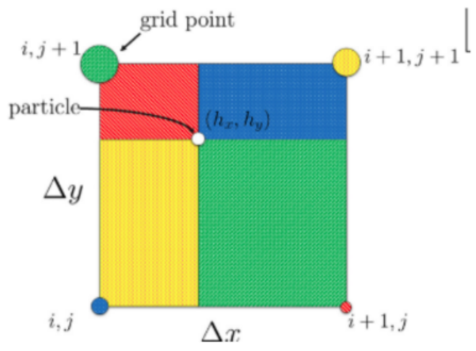

Figure 1: Point inside a grid cell distributing value to four nodes

5 Implementation Structure

The interpolation uses a particle-to-grid mapping approach. The structured grid is stored in contiguous memory and accessed using a one-dimensional index:

$$\text{index} = j \times GRID\_X + i$$

where i and j are the grid indices in the x and y directions. This storage layout improves memory locality and cache performance.

For each scattered point, the algorithm performs the following steps:

1. The grid cell containing the particle is identified from its coordinates. The indices i and j represent the column and row of the grid cell in which the particle lies:

$$i = \left\lfloor \frac{x}{dx} \right\rfloor, \quad j = \left\lfloor \frac{y}{dy} \right\rfloor$$

   Here, dx and dy are the grid spacings, i.e., the width and height of each grid cell.

2. Boundary checks ensure that the computed indices remain within grid limits.

3. The exact position of the particle inside the grid cell is computed using:

$$l_x = x - i \cdot dx, \quad l_y = y - j \cdot dy$$

   The terms $i \cdot dx$ and $j \cdot dy$ give the coordinates of the lower-left corner of the grid cell. Subtracting these values from (x, y) gives the particle's distance from the cell corner.

4. Distances from the opposite cell edges are computed:

$$dx - l_x, \quad dy - l_y$$

   These represent how far the particle is from the right and top edges of the cell.

5. Bilinear interpolation weights are computed using these distances.

6. The particle's value is distributed to the four surrounding grid nodes according to the computed weights.

Since grid values are accumulated directly in memory, contributions from multiple particles are combined efficiently.

This procedure is repeated for all particles and for each iteration of input data.

6 Analysis and Results

6.1 Theoretical Time Complexity

Let N be the number of scattered points.

For each particle, the algorithm identifies the grid cell, computes interpolation weights, and updates four grid nodes. Since the work per particle is constant, the overall time complexity is:

$$O(N)$$

Even when repeated for multiple iterations, the scaling remains linear.

6.2 Arithmetic Operations per Particle

For each scattered point, the arithmetic work can be described step by step:

- **Index computation:** 2 multiplications are used to determine the grid cell indices ($i = x \cdot inv\_dx$, $j = y \cdot inv\_dy$).
- **Cell boundary coordinates:** 2 multiplications compute the exact grid boundary location ($i \cdot dx$, $j \cdot dy$), where dx and dy are grid spacings.
- **Distance inside the cell:** 2 subtractions compute the particle's position inside the cell ($l_x = x - i \cdot dx$, $l_y = y - j \cdot dy$).
- **Remaining distances:** 2 subtractions compute distances from the opposite cell edges ($dx - l_x$, $dy - l_y$).
- **Weight computation:** 4 multiplications evaluate the bilinear interpolation weights.
- **Grid accumulation:** 4 additions update the four surrounding grid nodes.

$$8 \text{ multiplications} + 4 \text{ subtractions} + 4 \text{ additions}$$

$$\boxed{\approx 16 \text{ floating point operations per particle}}$$

Here, i and j denote the grid cell indices, while dx and dy represent the grid spacing. The terms $i \cdot dx$ and $j \cdot dy$ give the coordinates of the cell corner used to compute the particle's relative position within the cell.

6.3 Memory Access per Particle

For each particle, the interpolation routine performs the following memory operations:

- 2 reads for particle coordinates (x, y)
- 4 writes to grid nodes

Assuming double precision values (8 bytes each):

$$\text{Read traffic} = 2 \times 8 = 16 \text{ bytes}$$

$$\text{Write traffic} = 4 \times 8 = 32 \text{ bytes}$$

$$\boxed{\text{Total memory traffic} \approx 48 \text{ bytes per particle}}$$

Note that grid values may first be loaded into cache before being updated, but the dominant data movement remains proportional to these memory accesses.

6.4 Code Balance

Code balance is defined as the amount of memory traffic per floating point operation:

$$\text{Code Balance} = \frac{\text{Memory Traffic}}{\text{FLOPs}}$$

Using the computed values:

$$= \frac{48 \text{ bytes}}{16 \text{ FLOPs}} = 3 \text{ bytes/FLOP}$$

This relatively high code balance indicates that the algorithm is **memory-bandwidth bound**. Performance is therefore limited more by memory transfer speed than by computational throughput.

6.5 Cache Behavior

Because the grid is stored in contiguous memory, spatial locality is preserved. The four updated grid nodes are adjacent in memory, allowing efficient cache utilization. When nearby particles fall in nearby cells, cached grid values can be reused.

If particles are randomly distributed, updates occur across distant memory locations, reducing cache reuse and increasing cache misses. Since the computation per particle is small but memory updates are frequent, the algorithm is primarily memory-bandwidth bound.

6.6 Performance Comparison and Observations

6.6.1 Benchmark

Case	$N_x \times N_y$	Number of Points	Iterations
(a)	250 × 100	0.9 million	10
(b)	250 × 100	5 million	10
(c)	500 × 200	3.6 million	10
(d)	500 × 200	20 million	10
(e)	1000 × 400	14 million	10

These configurations were chosen to study the impact of increasing particle density and grid resolution on execution time and memory behavior.

6.6.2 Results

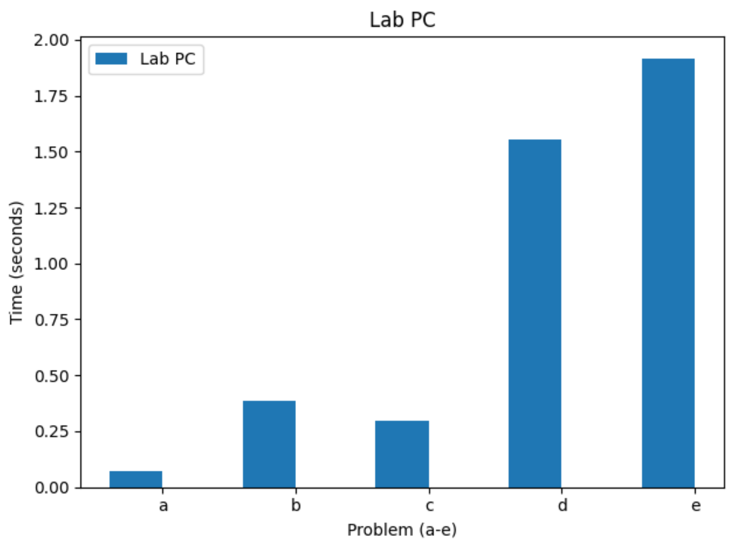

Figure 2: Time Taken for different problems by Lab's PC

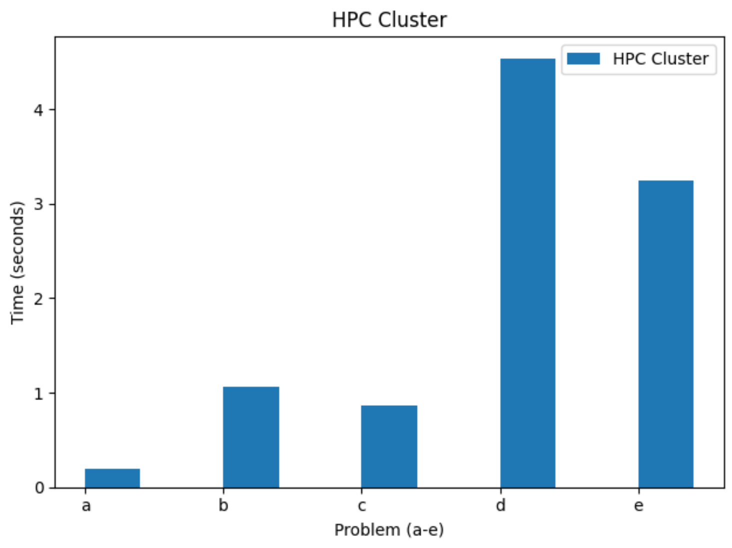

Figure 3: Time Taken for different problems by HPC Cluster

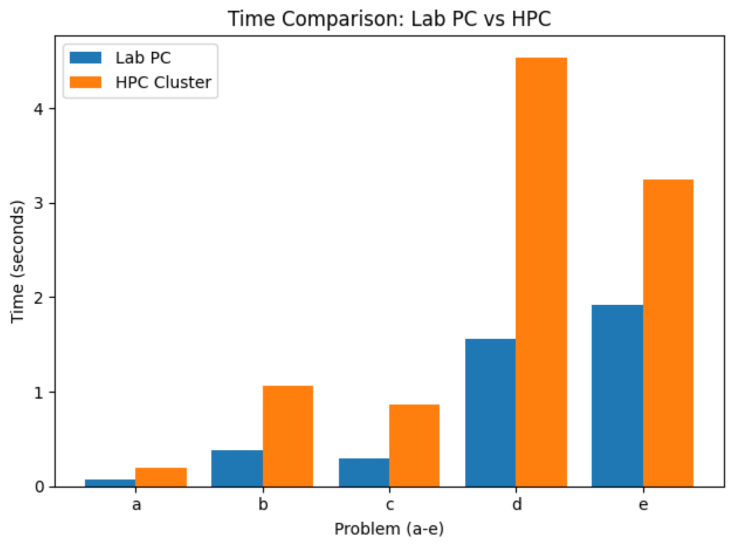

Figure 4: Comparison between Lab's PC and HPC cluster

6.6.3 Observation

- Execution time increases with the number of particles because the algorithm performs a fixed amount of work per particle.
- Larger grids also increase memory traffic and data movement.