CS 301
High-Performance Computing

Lab 03 - Serial Interpolation

Visha Sitapara (202301414)
Dhruv Patel (202301095)

February 21, 2026

Contents

1 Introduction

This lab focuses on implementing a serial bilinear interpolation algorithm to map scattered 2D data points onto a structured mesh grid. The objective is to ensure correctness first and then optimize performance through careful memory layout, indexing, and computational efficiency. The assignment emphasizes both numerical accuracy and performance evaluation on both lab PCs and Cluster.

2 Hardware Details

2.1 Hardware Details for LAB207 PCs

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Address sizes: 46 bits physical, 48 bits virtual
- Byte Order: Little Endian
- CPU(s): 12
- On-line CPU(s) list: 0-11
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 1
- NUMA node(s): 1
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 151
- Model name: 12th Gen Intel(R) Core(TM) i5-12500
- Stepping: 5
- CPU max MHz: 4600.0000
- CPU min MHz: 800.0000
- BogoMIPS: 5990.40
- Virtualization: VT-x
- L1d cache: 288K
- L1i cache: 192K

- L2 cache: 7.5M
- L3 cache: 18M
- NUMA node0 CPU(s): 0-11
- Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm epb invpcid_single tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2 erms invpcid xsaveopt dtherm ida arat pln pts

2.2 Hardware Details for HPC Cluster (Node gics0)

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Byte Order: Little Endian
- CPU(s): 24
- On-line CPU(s) list: 0-23
- Thread(s) per core: 2
- Core(s) per socket: 6
- Socket(s): 2
- NUMA node(s): 2
- Vendor ID: GenuineIntel
- CPU family: 6
- Model: 63
- Model name: Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz
- Stepping: 2
- CPU MHz: 2907.656
- BogoMIPS: 4804.69
- Virtualization: VT-x
- L1d cache: 32K
- L1i cache: 32K

- L2 cache: 256K
- L3 cache: 15360K
- NUMA node0 CPU(s): 0-5, 12-17
- NUMA node1 CPU(s): 6-11, 18-23

3 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) represent spatial coordinates and f_i are known function values. For this assignment, assume:

$$f_i = 1$$

The objective is to interpolate these scattered values onto a structured mesh grid of size $M \times M$. The structured mesh consists of regularly spaced grid points:

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

where Δx and Δy define uniform grid spacing:

$$\Delta x = \frac{X_{\max}}{M}, \quad \Delta y = \frac{Y_{\max}}{M}$$

All scattered points lie within a domain of size 1×1, i.e.,

$$0 \leq x_i \leq X_{\max}, \quad 0 \leq y_i \leq Y_{\max}$$

The goal is to compute interpolated values $F(X_i, Y_j)$ on the structured grid using linear interpolation.

4 Benchmarking Methodology

Input Data Configurations

Input data is generated using `input_fileMaker.c` with the following configurations:

(a) $N_x = 250$, $N_y = 100$, Number of points $= 0.9 \times 10^6$, Maxiter $= 10$

(b) $N_x = 250$, $N_y = 100$, Number of points $= 5 \times 10^6$, Maxiter $= 10$

(c) $N_x = 500$, $N_y = 200$, Number of points $= 3.6 \times 10^6$, Maxiter $= 10$

(d) $N_x = 500$, $N_y = 200$, Number of points $= 20 \times 10^6$, Maxiter $= 10$

(e) $N_x = 1000$, $N_y = 400$, Number of points $= 14 \times 10^6$, Maxiter $= 10$

5 Graphical Results

5.1 Bar plot of execution time vs. problem index

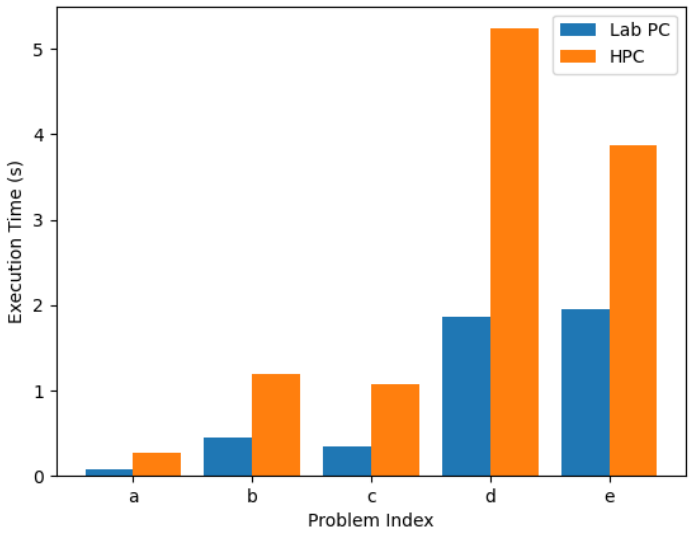
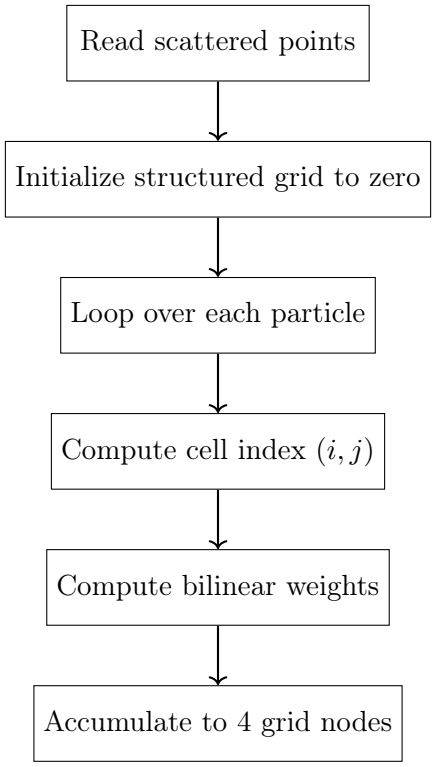


Figure 1: Execution time vs. Problem index (a to e) for both Lab PC and HPC Cluster

6 Implementation Approach

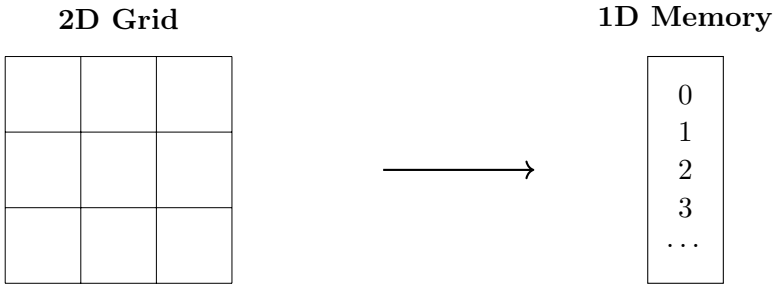
6.1 Interpolation Workflow



6.2 Data Layout in Memory

6.2.1 Structured Grid Representation

Although conceptually the grid is two-dimensional: $G = \{(X_i, Y_j)\}$; it is stored in **row-major 1D format**.



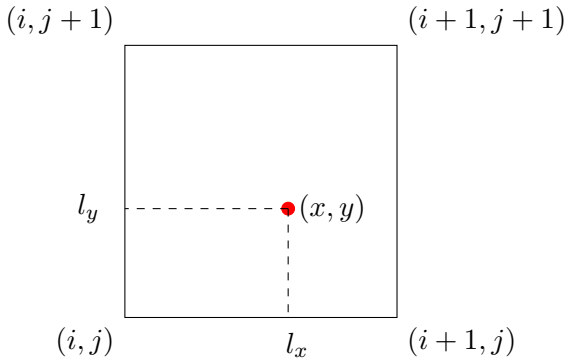
The mapping is defined as:

$$\text{index} = j \times N_x + i$$

This row-major ordering ensures contiguous memory access along rows.

6.2.2 Scattered Point Inside a Grid Cell

Each scattered point is stored as (x_i, y_i). The scattered point distributes its value to the four surrounding grid nodes based on its relative position inside the cell.



The memory layout is:

```
| x0 | y0 | x1 | y1 | x2 | y2 | ... | xN | yN |
```

The points are accessed sequentially, which ensures:

- Good spatial locality
- Cache-friendly streaming access

6.3 Grid Indexing Strategy

Given a scattered point (x, y) in a domain $[0, 1] \times [0, 1]$:

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

The containing grid cell indices are computed as:

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

Boundary protection ensures:

$$0 \leq i \leq N_x - 1$$
$$0 \leq j \leq N_y - 1$$

The lower-left corner of the cell is:

$$X_i = i\Delta x, \quad Y_j = j\Delta y$$

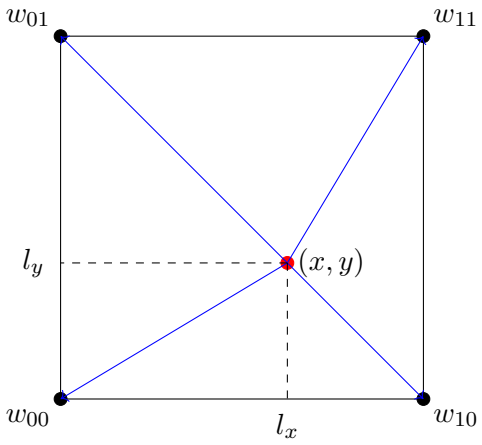
6.4 Local Distance Computation

$$l_x = x - X_i$$

$$l_y = y - Y_j$$

These represent the relative position of the scattered point inside the grid cell.

6.5 Bilinear Weight Distribution



Each scattered point contributes to exactly four grid nodes using bilinear weights. The four interpolation weights are:

$$w_{00} = (\Delta x - l_x)(\Delta y - l_y)$$

$$w_{10} = l_x(\Delta y - l_y)$$

$$w_{01} = (\Delta x - l_x)l_y$$

$$w_{11} = l_x l_y$$

These weights represent the relative area contributions inside the cell.

6.6 Mapping Scattered Points to Grid Nodes

Each scattered point distributes its value to the four nearest grid nodes:

$$F(i, j) \mathrel{+}= w_{00} f$$

$$F(i + 1, j) \mathrel{+}= w_{10} f$$

$$F(i, j + 1) \mathrel{+}= w_{01} f$$

$$F(i+1, j+1) \mathrel{+}= w_{11} f$$

Thus, each particle affects only **four grid nodes**.

7 Analysis Tasks

Q1. **Compute the theoretical time complexity.**

In the particle-based interpolation approach, the algorithm loops over all scattered points. Each of the N scattered points requires constant time work, the total time complexity is:

$$T(N) = O(N)$$

If the interpolation is performed for Maxiter iterations, the overall complexity becomes:

$$T(N) = O(N \cdot \text{Maxiter})$$

Q2. **Estimate arithmetic operations per particle.**

For each particle (x_i, y_i, f_i), the operations include:

Index computation (2 divisions):

$$i = \left\lfloor \frac{x_i}{\Delta x} \right\rfloor, \quad j = \left\lfloor \frac{y_i}{\Delta y} \right\rfloor$$

Grid reconstruction (2 multiplications):

$$X_i = i\Delta x, \quad Y_j = j\Delta y$$

Local distances (2 subtractions):

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

Weight computations (8 subtractions + 4 multiplications):

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

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

Grid accumulation (4 multiplications + 4 additions):

$$F \mathrel{+}= w \cdot f_i$$

Total arithmetic cost per particle $\approx \boxed{26}$ floating-point operations.

Q3. **Perform memory access analysis and Analyze cache behavior.**

For each particle, the algorithm performs:

- 3 reads from scattered data: (x_i, y_i, f_i)
- 4 read-modify-write updates to grid values:

$$F_{i,j},\ F_{i+1,j},\ F_{i,j+1},\ F_{i+1,j+1}$$

Cache Behavior:

- Scattered point array exhibits strong spatial locality and high cache efficiency.
- Grid updates may cause cache misses due to irregular access patterns.
- If multiple nearby particles fall in the same cell, temporal locality improves.
- For large grids (when M^2 exceeds cache size), capacity misses increase.

Q4. **Suggest further optimizations under better hardware assumptions.**

Under improved hardware, the following optimizations can be applied:

- **Data Layout Optimization:** Use Structure of Arrays (SoA) instead of Array of Structures (AoS) to improve cache-line utilization and enable vectorization.
- **SIMD Vectorization:** Process multiple particles simultaneously to reduce per-particle computation time.
- **Parallelization:** Use OpenMP or MPI to distribute particles across cores or nodes for improved scalability.

Q5. **Measure and compare the execution time, plot a bar chart and analyze the observed performance differences based on hardware characteristics and memory behavior.**

For bar plot, refer to Figure 1.

Execution time increases nearly linearly with problem size on both systems; however, the HPC cluster shows higher runtime than the Lab PC for this serial implementation. This may be due to:

- Higher memory latency or lower single-core frequency on the HPC node.
- The code being purely serial, thus not utilizing the multi-core advantage of the cluster.
- Memory-bound behavior of the interpolation algorithm, where frequent grid updates lead to cache misses and reduced cache efficiency.

Q6. **Explain your implementation approach pictorially.**

Refer to Section 6.

8 Conclusion

- The serial bilinear interpolation algorithm was successfully implemented and validated for correctness across all configurations.
- The theoretical time complexity $O(N \cdot \text{Maxiter})$ was confirmed experimentally, showing near-linear scaling with the number of particles.
- Performance analysis revealed that the algorithm is memory-bound due to frequent grid read-modify-write operations.
- The Lab PC outperformed the HPC cluster for the serial version, highlighting the impact of single-core frequency and memory latency.