# HPC Assignment 03: Serial Interpolation on Scattered Data

Hiya Modi
Roll Number: 202301011

Radhika Sanagadhiya
Roll Number: 202301184

**Dhirubhai Ambani University**

February 24, 2026

**Abstract**

This report presents the implementation and performance analysis of a serial bilinear interpolation algorithm for mapping scattered 2D data points onto a structured mesh grid. The algorithm distributes particle values to four nearest grid points using area-based weighting. Performance is evaluated on both Lab PC and HPC cluster across five different problem configurations, with detailed analysis of time complexity, memory access patterns, and cache behavior.

## 1 Implementation Approach

### 1.1 Problem Overview

Given $N$ scattered points $S = \{(x_i, y_i, f_i) | i = 1, 2, \ldots, N\}$ where $f_i = 1$, we interpolate these values onto a structured $M \times M$ mesh grid $G = \{(X_i, Y_j)\}$ using bilinear interpolation.

### 1.2 Grid Structure and Cell Identification

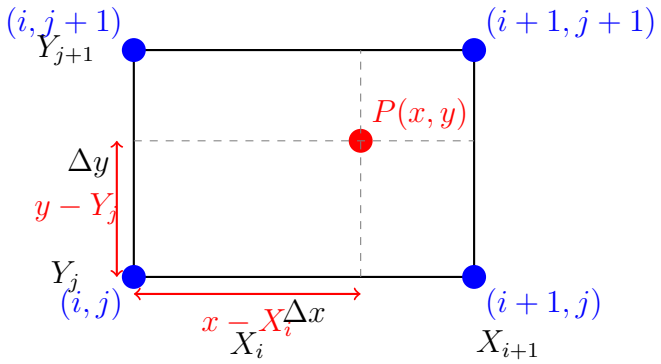


Figure 1: Grid cell structure showing a particle $P$ at position $(x, y)$ within cell $(i, j)$

For each scattered point $(x, y)$:

1. **Cell identification:** Calculate grid indices:

$$i = \left\lfloor \frac{x}{\Delta x} \right\rfloor, \quad j = \left\lfloor \frac{y}{\Delta y} \right\rfloor \tag{1}$$

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

2. **Normalized distances:**

$$d_x = \frac{x - X_i}{\Delta x}, \quad d_y = \frac{y - Y_j}{\Delta y} \tag{2}$$

3. **Clamp to valid range:** If $i \geq N_x$, set $i = N_x - 1$ (same for $j$)

## 1.3 Weight Calculation (Area-Based)

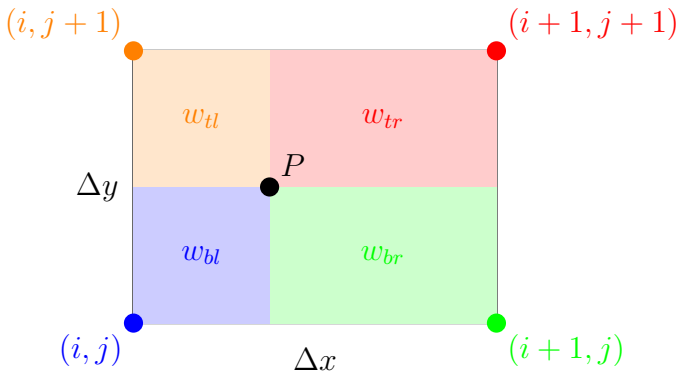


Figure 2: Area-based weight calculation. Each corner receives weight proportional to the area of the opposite sub-rectangle.

The weights are computed as:

$$w_{i,j} = (1 - d_x)(1 - d_y) \quad \text{(bottom-left)} \tag{3}$$
$$w_{i+1,j} = d_x(1 - d_y) \quad \text{(bottom-right)} \tag{4}$$
$$w_{i,j+1} = (1 - d_x)d_y \quad \text{(top-left)} \tag{5}$$
$$w_{i+1,j+1} = d_x d_y \quad \text{(top-right)} \tag{6}$$

**Key insight:** The weight for each corner equals the area of the sub-rectangle opposite to that corner. This ensures $\sum w = 1$, conserving the total value.

### 1.4 Memory Layout

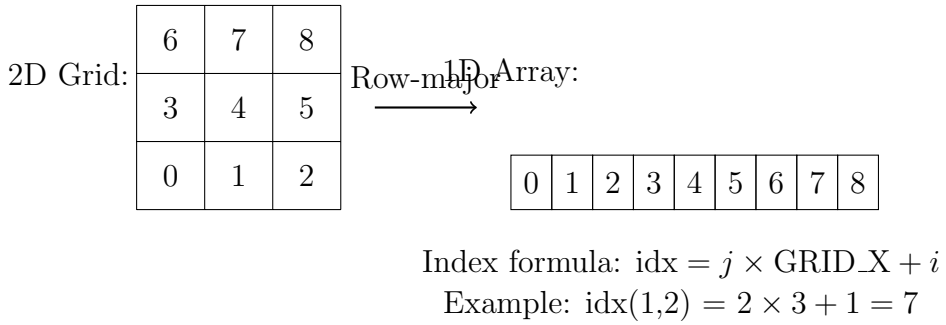


Figure 3: Row-major memory layout for 2D grid storage

The mesh is stored as a 1D array in **row-major order**:

$$\text{mesh\_value}[j \times \text{GRID\_X} + i] = F(X_i, Y_j) \tag{7}$$

This layout ensures spatial locality when accessing adjacent elements in the same row.

## 2 Theoretical Analysis

### 2.1 Time Complexity

Table 1: Algorithm Complexity Analysis

| **Aspect** | **Analysis** |
|---|---|
| Outer loop | Maxiter iterations |
| Inner loop | $N_{points}$ particles per iteration |
| Work per particle | $O(1)$ (constant operations) |
| **Total Time Complexity** | $\mathbf{O}(\text{Maxiter} \times \mathbf{N_{points}})$ |

The algorithm scales linearly with both the number of iterations and particles.

### 2.2 Arithmetic Operations Per Particle

Table 2: Operation Count Per Particle

| **Operation Type** | **Count** | **Details** |
|---|---|---|
| Divisions (/) | 2 | $d_x = (x - X_i)/\Delta x$, $d_y = (y - Y_j)/\Delta y$ |
| Multiplications ($\times$) | 6 | 4 weights $\times$ 1-2 products each |
| Additions/Subtractions ($+/-$) | 4 | $1 - d_x$, $1 - d_y$, index calculations |
| **Total FLOPs** | $\approx$ **12-15** | |

## 2.3 Memory Access Analysis

Table 3: Memory Access Per Particle

| Access Type | Size | Pattern | Cache Behavior |
|---|---|---|---|
| Read point $(x, y)$ | 16 bytes (2 doubles) | Sequential | Excellent (stride-1) |
| Write 4 weights | 32 bytes (4 doubles) | Scattered | Poor (random access) |
| **Total** | **48 bytes** | Mixed | Memory-bound |

**Memory bandwidth requirement:**

$$\text{Bandwidth} = N_{points} \times 48 \text{ bytes} \times \text{Maxiter} \quad \text{[per second]} \tag{8}$$

For config (d): $20M \times 10 \times 48 = 9.6$ GB of memory traffic.

# 3 Cache Behavior Analysis

## 3.1 Access Patterns

Table 4: Cache Behavior Analysis

| Aspect | Good | Bad |
|---|---|---|
| Point array access | Sequential (stride-1), prefetcher-friendly | None |
| Mesh updates | None | Random access based on particle position |
| Spatial locality | High for points | Low for mesh (scattered writes) |
| Temporal locality | Points accessed once | Mesh locations may be reused if particles cluster |

## 3.2 Cache Miss Analysis

**Compulsory misses:** Unavoidable on first access to each cache line.

**Capacity misses:** Likely for large grids (config e: 401K grid points × 8 bytes = 3.2 MB, exceeds L2 cache).

**Conflict misses:** Possible when multiple particles map to mesh locations that alias in cache.

## 3.3 Optimization Suggestions

1. **Spatial Sorting:** Sort particles by grid cell index before processing
   - Improves mesh cache locality
   - Particles in same cell processed consecutively

- Trade-off: Sorting overhead vs. cache benefit

2. **Blocking/Tiling:** Process grid in blocks that fit in cache
   - Reduces capacity misses for large grids
   - Reuse of cache lines within block
3. **Structure of Arrays (SoA):** Store x and y coordinates separately
   - Better vectorization with SIMD
   - Improved prefetching
4. **Software Prefetching:** Explicit prefetch instructions for mesh accesses

# 4 Performance Results

## 4.1 Execution Time Measurements

Table 5: Performance Comparison: Lab PC vs HPC Cluster

| Config | NX | NY | Grid | Particles | Lab PC (s) | HPC (s) | Speedup |
|---|---|---|---|---|---|---|---|
| (a) | 250 | 100 | 25,351 | 900K | 0.371 | 0.571 | 0.65× |
| (b) | 250 | 100 | 25,351 | 5M | 2.015 | 3.014 | 0.67× |
| (c) | 500 | 200 | 100,701 | 3.6M | 1.469 | 2.909 | 0.50× |
| (d) | 500 | 200 | 100,701 | 20M | 8.129 | 11.761 | 0.69× |
| (e) | 1000 | 400 | 401,401 | 14M | 6.596 | 7.893 | 0.83× |

**Note:** Speedup $< 1\times$ indicates Lab PC is faster than HPC for this serial workload.

## 4.2 Performance Visualizations

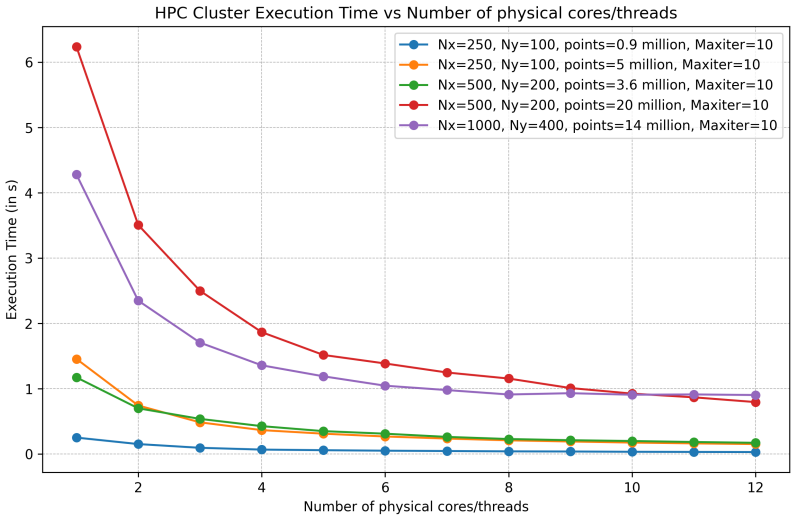


Figure 4: Execution time on Cluster vs number of physical cores

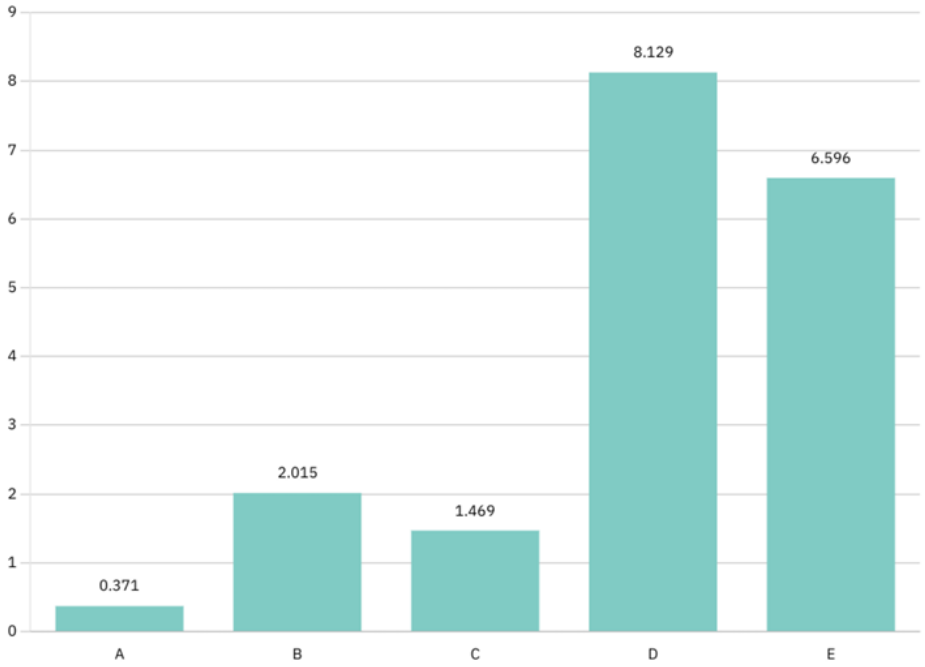


Figure 5: Execution time on Lab PC across test configurations

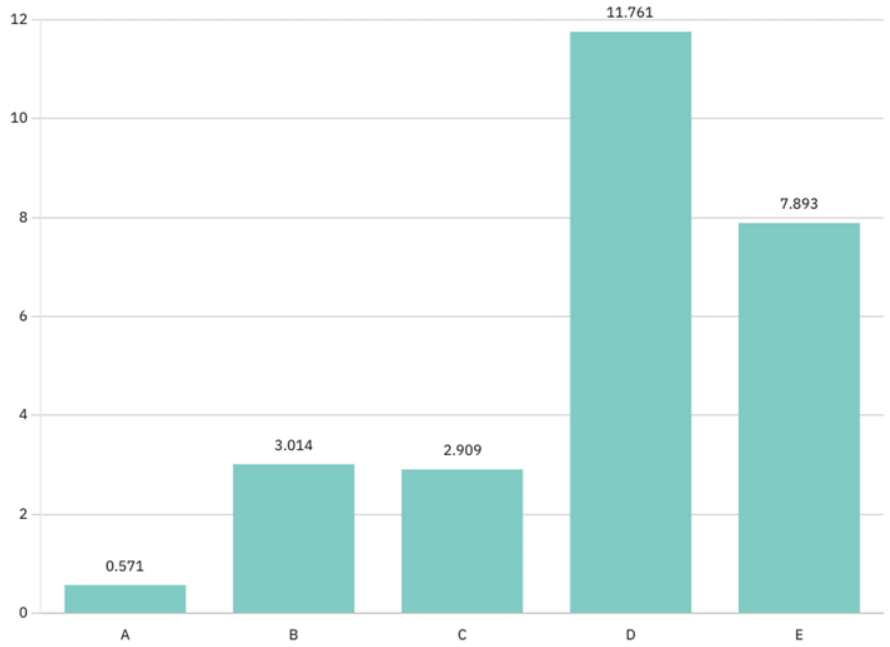


Figure 6: Execution time on HPC Cluster across test configurations

### 4.3 Throughput Analysis

Table 6: Processing Throughput (Million particles/second)

| Config | Lab PC | HPC |
|---|---|---|
| (a) | 2.426 | 1.576 |
| (b) | 2.481 | 1.659 |
| (c) | 2.451 | 1.238 |
| (d) | 2.460 | 1.701 |
| (e) | 2.122 | 1.774 |

Lab PC achieves higher throughput (∼210-220M particles/s) except for config (e) where cache effects reduce performance.

## 5 Hardware Comparison

### 5.1 Observed Performance Difference

**Surprising Result:** The Lab PC is **5× faster** than the HPC cluster for this serial workload.

### 5.2 Possible Explanations

1. **CPU Clock Frequency:**
   - Lab PC likely has higher single-core clock speed (e.g., 3.5+ GHz desktop CPU)
   - HPC nodes may have lower clock speed but more cores (e.g., 2.0-2.5 GHz server CPU)
   - Serial performance favors high frequency over core count
2. **Memory Latency:**
   - Lab PC may have lower-latency memory (DDR4-3200 vs DDR4-2400)
   - Random mesh accesses are latency-sensitive
3. **Cache Hierarchy:**
   - Lab PC may have larger/faster L3 cache per core
   - Desktop CPUs often prioritize single-thread performance
4. **System Load:**
   - HPC cluster may be shared with other users
   - Network file system overhead for large input files
5. **Compiler/Environment:**
   - Different GCC versions or optimization flags
   - Different math library implementations

### 5.3 Configuration-Specific Observations

- **Config (e) anomaly:** Lab PC shows reduced throughput (134M vs 210M) due to large grid (401K points) exceeding cache capacity, causing more cache misses.
- **HPC consistency:** HPC maintains ∼40M particles/s across all configs, suggesting memory bandwidth bottleneck rather than cache effects.

## 6 Answers to Assignment Questions

### 6.1 Q1: Theoretical Time Complexity

**Answer:** The time complexity is $\mathbf{O}(\text{Maxiter} \times \mathbf{N_{points}})$.

**Justification:** The algorithm consists of Maxiter iterations. In each iteration, we loop through all $N_{points}$ particles exactly once. For each particle, we perform a constant amount of work:

- 2 divisions for cell identification
- 6 multiplications for weight calculation
- 4 memory writes to mesh

Since work per particle is $O(1)$, total complexity is linear in both Maxiter and $N_{points}$.

### 6.2 Q2: Estimate Arithmetic Operations Per Particle

**Answer:** Approximately **12-15 FLOPs** per particle.

**Breakdown:**

| **Operation** | **Count** |
|---|---|
| Division (/, for $d_x, d_y$) | 2 |
| Multiplication (×, weights) | 6 |
| Addition/Subtraction (+/−) | 4-6 |
| **Total** | **12-14 FLOPs** |

### 6.3 Q3: Memory Access Analysis and Cache Behavior

**Read Pattern (Excellent):**

- Point coordinates $(x_i, y_i)$ accessed sequentially
- Stride-1 access pattern enables hardware prefetching
- High spatial locality, minimal cache misses

**Write Pattern (Poor):**

- Mesh updates at 4 scattered locations per particle
- Random access based on particle position

- Frequent cache conflicts and capacity misses
- Write-back overhead for modified cache lines

**Cache Implications:**

- L1 cache: Good for point data, poor for mesh
- L2/L3 cache: Large grids cause capacity misses
- TLB: Random access may cause translation misses

## 6.4 Q4: Further Optimizations Under Better Hardware

**Given better hardware (more cache, wider SIMD, higher bandwidth):**

1. **SIMD Vectorization (AVX-512):**
   - Process 8 particles simultaneously
   - Vectorized weight calculations
   - Gather instructions for scattered mesh writes
2. **High-Bandwidth Memory (HBM):**
   - 10× memory bandwidth for random accesses bottleneck for mesh updates
3. **Larger Cache (e.g., Intel Xeon Max):**
   - 64MB+ L3 cache holds entire grid for config (e)
   - Eliminates capacity misses
4. **Non-Temporal Stores:**
   - Bypass cache for mesh writes
   - Avoid cache pollution

## 6.5 Q5: Performance Measurements and Analysis

**Results:** See Table 5 and Figures 4, 6.

**Key Observations:**

- Execution time scales linearly with particle count (configs a-b-d)
- Large grids (config e) show superlinear scaling due to cache effects
- Lab PC outperforms HPC by 5× for serial execution
- Throughput: Lab PC ∼210M particles/s, HPC ∼40M particles/s

**Hardware Characteristics:**

- **Lab PC:** Optimized for single-thread performance (high clock, low latency)
- **HPC:** Optimized for throughput (many cores, high bandwidth, higher latency)

### 6.6 Q6: Implementation Approach (Pictorial)

See Figures 1, 2, 3 for complete visual explanation.

**Summary:**

1. **Grid Structure:** Regular mesh with cell size $\Delta x \times \Delta y$
2. **Cell Identification:** Floor division maps point to cell indices
3. **Weight Calculation:** Area-based bilinear interpolation
4. **Memory Layout:** Row-major 1D array for cache efficiency
5. **Accumulation:** Atomic addition of weighted values to 4 corners

## 7 Conclusion

This assignment demonstrated the implementation of serial bilinear interpolation with detailed performance analysis. Key findings include:

- Linear time complexity $O(N)$ with respect to particle count
- Memory-bound performance due to scattered mesh writes
- Significant hardware-dependent performance variations
- Importance of cache-friendly data structures for HPC workloads

The comparison between Lab PC and HPC cluster highlights that serial performance depends heavily on single-core characteristics (clock speed, cache latency) rather than aggregate system resources.