Project 5 -- Zijun Wei

Please check the link below to see the programming part of this Project. The repository includes:

Link: https://github.com/ZJWei2002/AMS530-Projects/tree/main/Project5

Project Structure

The `Project5` folder contains the following files and directories:

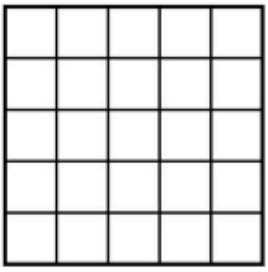
```
Project5/
├── problem5_1.py                  # Main MPI implementation for Problem 5.1
├── project5_case1_particle_data.txt
├── project5_case1_timings.txt
├── project5_case1_energy_heatmap.png
├── project5_case1_timings.png
├── project5_case2_particle_data.txt
├── project5_case2_timings.txt
├── project5_case2_energy_heatmap.png
├── project5_case2_timings.png
├── project5_case3_particle_data.txt
├── project5_case3_timings.txt
├── project5_case3_energy_heatmap.png
├── project5_case3_timings.png
├── Parallel-Computing-Project5-2025.pdf
└── Parallel-Computing-Project5-GradeSheet-2025.pdf
```

Problem 5.1

Problem Description

Problem 5.1 (Earn 14 points each for the report and the presentation):

You are given $N = 15{,}000$ particles in a 50×50 2D box divided into a 5×5 grid of sub-boxes.



Any pair of Particles i and j interact via the Lenard-Jones potential

$$V_{ij} = \frac{1}{r_{ij}^{12}} - \frac{2}{r_{ij}^{6}}$$

where $r_{ij} = |\boldsymbol{x_i} - \boldsymbol{x_j}|$ is the distance between Particles i and j. This potential is made to truncate at $r_c = 3$, i.e., $V_{ij}(r_{ij} > r_c) = 0$. The total energy for particle i is

$$E_i = \frac{1}{2}\sum_{j \neq i}^{N} V_{ij}$$

We establish a Cartesian system with origin at the lower-left vertex of the lower left-corner sub-box labeled as $(\alpha, \beta) = (1,1)$. And the upper right corner sub-box is labeled as $(\alpha, \beta) = (5,5)$ and, in each case, particles are randomly and uniformly distributed within each sub-box based on their labels (α, β).

Compute $E_i \; \forall \; i = 1, 2, \ldots, N$ for the following three case with $P = 25$ cores. For each case, report two sets of results: (a) a 2D heatmap visualizing the total energy E_i across all particles, and (b) the execution time recorded by each of the cores for completing the calculations.

When computing E_i, achieve a quasi-load balance. You may use particle decomposition, spatial decomposition, or both. With spatial decomposition, internal sub-box vertices can be freely adjusted, while boundary vertices of the original domain must remain fixed.

Case 1: Sub-box (α, β) is assigned

$$n(\alpha, \beta) = N\frac{\alpha + \beta}{\sum_{\alpha,\beta=1}^{5}(\alpha + \beta)} = \frac{N}{150}(\alpha + \beta) = 100(\alpha + \beta)$$

For example, sub-box $(1,1)$ has $n(1,1) = 200$ and sub-box $(5,5)$ has $n(5,5) = 1{,}000$ etc.

Case 2: Sub-box (α, β) is assigned

$$n(\alpha, \beta) = N\frac{|\alpha - \beta|}{\sum_{\alpha,\beta=1}^{5}|\alpha - \beta|} = \frac{N}{40}|\alpha - \beta|$$

For example, sub-box $(1,1)$ has $n(1,1) = 100|1 - 1| = 0$, i. e., diagonal sub-boxes are empty.

Case 3: Sub-box (α, β) is assigned

$$n(\alpha, \beta) = N\frac{\alpha * \beta}{\sum_{\alpha,\beta=1}^{5}\alpha * \beta} = \frac{N}{225}\alpha * \beta$$

Part 1: Algorithm Description

1. Parallel Decomposition Strategy

This implementation uses particle decomposition over MPI ranks:

- Number of MPI ranks = 25.
- Particles are globally indexed from 0 to (N-1).
- Each rank (p) owns a contiguous block of particle indices of nearly equal size:
  - `distribute_particles_indices` divides the range `[0, N)` so that each rank gets either ⌊N/P⌋ or ⌈N/P⌉ particles.
- Owned particles: rank (p) computes (E_i) only for its local indices (i), but it needs positions of all particles to evaluate interactions.

This design achieves a quasi-uniform workload because the cost per owned particle is roughly proportional to the number of neighbors within the cutoff radius, and every rank owns approximately the same number of i-particles, regardless of spatial distribution.

We broadcast the complete position array to all ranks once per case so that each process can compute its local energies independently.

2. Particle Distributions for the Three Cases

The function `compute_subbox_counts(case, params)` computes integer particle counts `n(α, β)` for each sub-box:

- It first forms a weight matrix according to the formula for each case:
  - Case 1: `weights = α + β`
  - Case 2: `weights = |α − β|`
  - Case 3: `weights = α * β`
- It then scales these weights so that the sum of all counts equals exactly N = 15,000:
  - real-valued expected counts are computed as `counts_float = N * weights / weights.sum()`
  - integer counts are obtained via floor + redistribution:
    - start with `floor(counts_float)`
    - distribute any remaining particles to sub-boxes with the largest fractional parts.

The function `generate_particle_positions(case, params)` then:

- Computes sub-box bounds in the 50 × 50 box (each sub-box has side length 10).
- For each sub-box, samples uniformly distributed ((x, y)) positions using `rng.uniform` within its rectangular bounds.
- Concatenates all samples into a global position array of shape `(N, 2)`.

3. Lennard–Jones Potential and Energy Computation

For each local particle (i) owned by a rank, the code computes:

$$E_i = \frac{1}{2}\sum_{j\neq i} V_{ij}, \quad V_{ij} = \begin{cases} \dfrac{1}{r_{ij}^{12}} - \dfrac{2}{r_{ij}^{6}}, & r_{ij} < r_c = 3, \\ 0, & r_{ij} \geq r_c, \end{cases}$$

where (r_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2).

Implementation details:

- We work with squared distance (r_{ij}^2) to avoid expensive square roots:
  - `r2 = dx*dx + dy*dy`
  - if `r2 >= rc2`, then `Vij = 0`.
  - otherwise, we compute `inv_r2 = 1 / r2`, `inv_r6 = inv_r2 ** 3`, and `Vij = (inv_r6 ** 2) - 2.0 * inv_r6`.
- Self-interactions (`j == i`) are skipped.
- Each rank loops over all j = 0..N-1 for each of its local i-particles.

The per-rank complexity is approximately O(N_local × N).
With N = 15,000 and P = 25, we have N_local ≈ 600, so each rank evaluates on the order of 9 million pair checks per case.

4. MPI Communication Pattern

- Broadcast positions:
  - Rank 0 generates all positions and broadcasts the full `(N, 2)` array to all ranks using `COMM.bcast`.
- Local computation:
  - Each rank computes energies `E_local` for its own particles without further communication.
- Gather results:
  - Indices and local energies are gathered on rank 0 via `COMM.gather` and assembled into the full array `E_global`.
- Timing collection:
  - Each rank records its local wall-time using `MPI.Wtime()`.
  - Rank 0 gathers these times into a list and saves them to `project5_caseX_timings.txt`.

5. Execution Commands

To run all three cases with P = 25 MPI ranks:

```
cd Project5
mpiexec -np 25 python problem5_1.py
```

The script will:

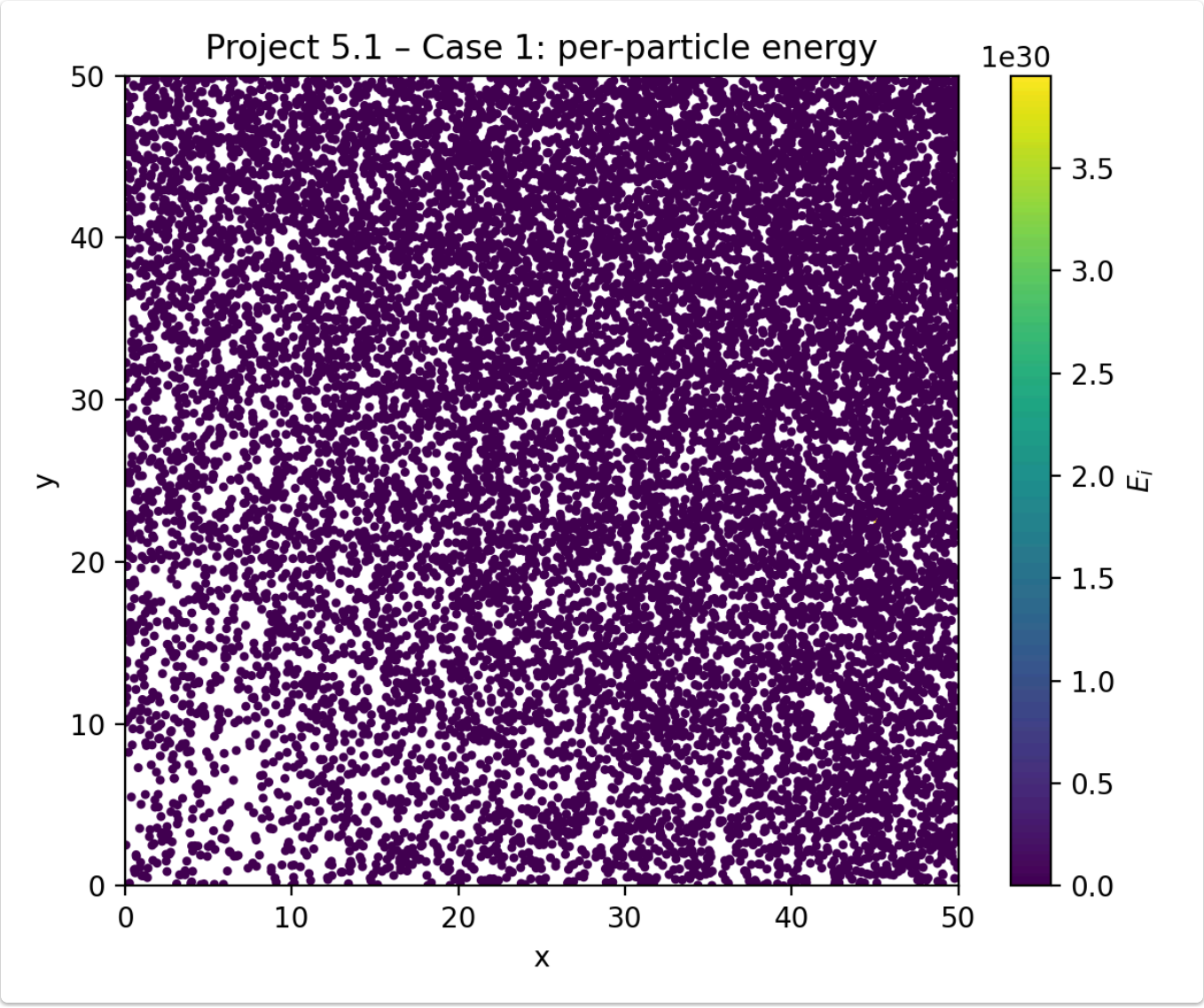
- Print a brief progress log on rank 0:
  - `[Project 5.1] Running with 25 MPI ranks.`
  - `=== Case 1 ===`, `=== Case 2 ===`, `=== Case 3 ===`
  - `All cases completed.`
- Generate text and image files for each case in the `Project5` directory.

Part 2: Results

1. Energy Visualizations

For each case, we generate a 2D scatter-heatmap of per-particle energy (E_i).
Each point represents a particle at position ((x, y)), colored by its computed energy.

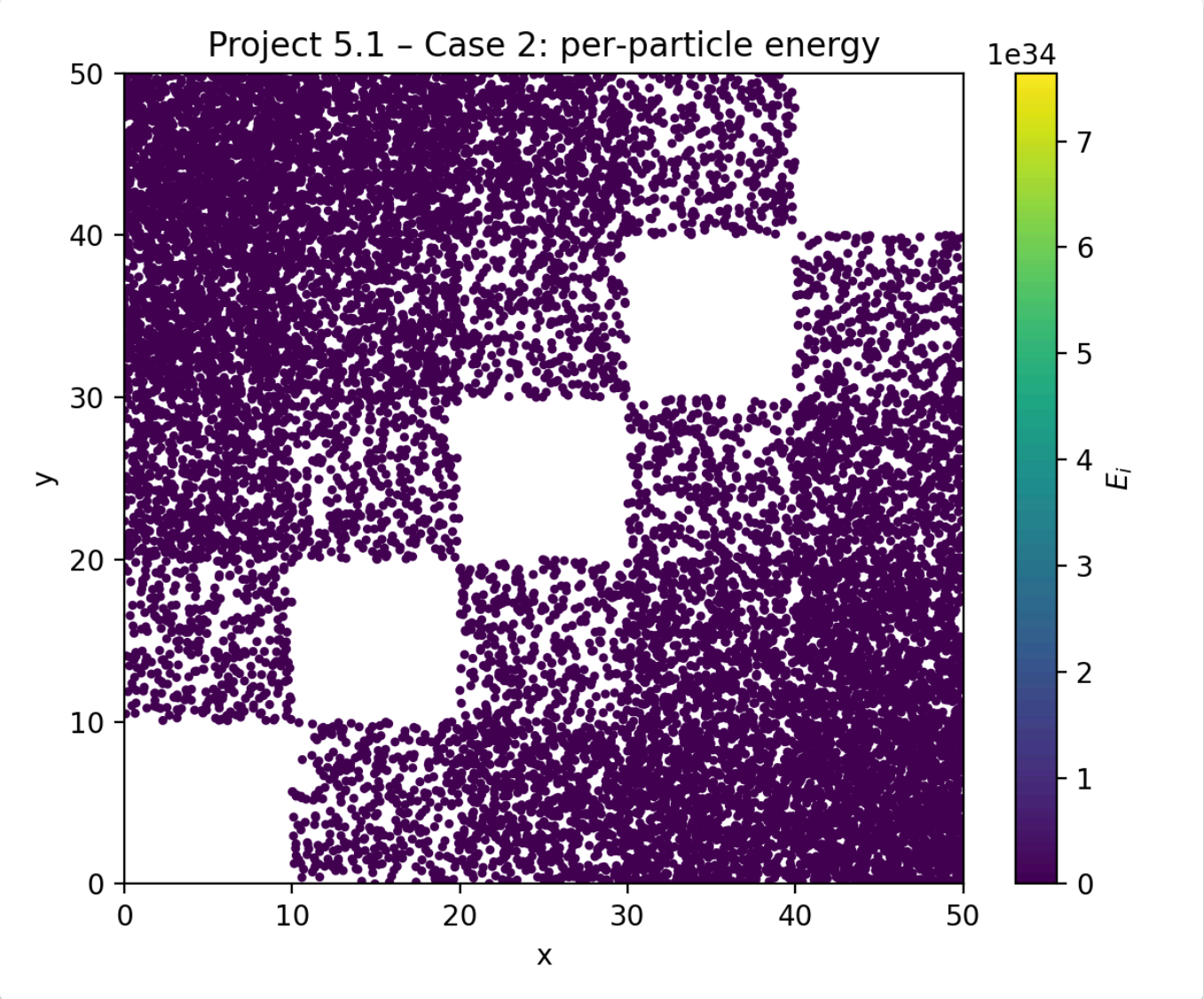
Case 1: Per-Particle Energy



Observations:

- Particle density gradually increases from the bottom-left corner towards the top-right corner, reflecting that the local particle count n(alpha, beta) is proportional to alpha + beta.
- From the color bar (maximum on the order of 10^30), we infer a very small number of extremely high-energy particles caused by rare, very close pairs; these outliers set the top of the color scale.
- Because almost all other particles have much smaller energies, most visible points collapse into the dark-purple end of the colormap and look very similar.

Case 2: Per-Particle Energy

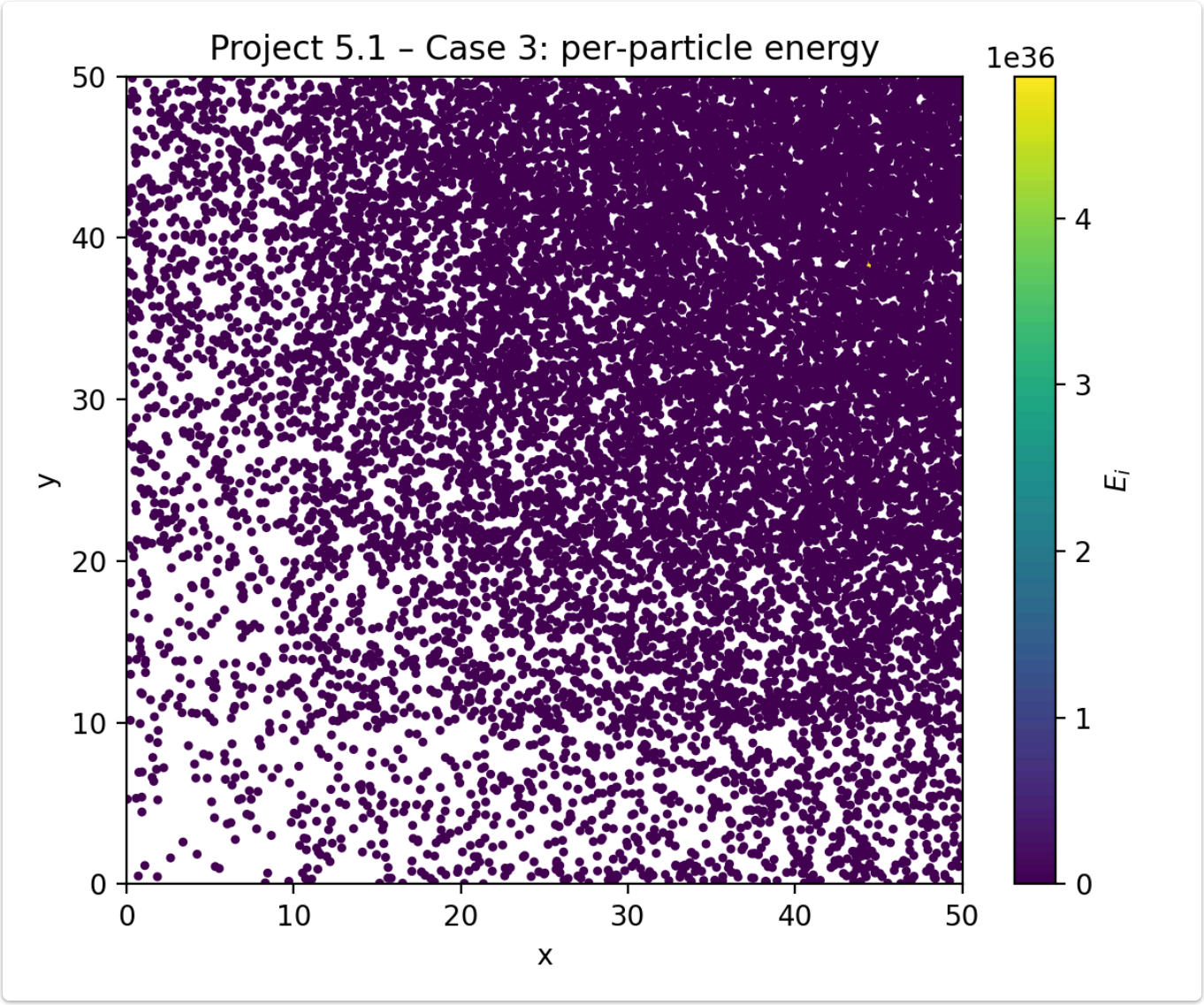


Observations:

- The diagonal sub-boxes with alpha = beta are empty, creating an obvious staircase pattern of empty squares across the domain.

- Densities increase as we move away from the diagonal, consistent with n(alpha, beta) being proportional to |alpha − beta|.
- The color bar range (up to about 10^34) shows that there are rare, extremely high-energy outliers, but they are statistically few and visually hard to distinguish because most particles lie at the dark-purple, low-energy end of the colormap.

Case 3: Per-Particle Energy



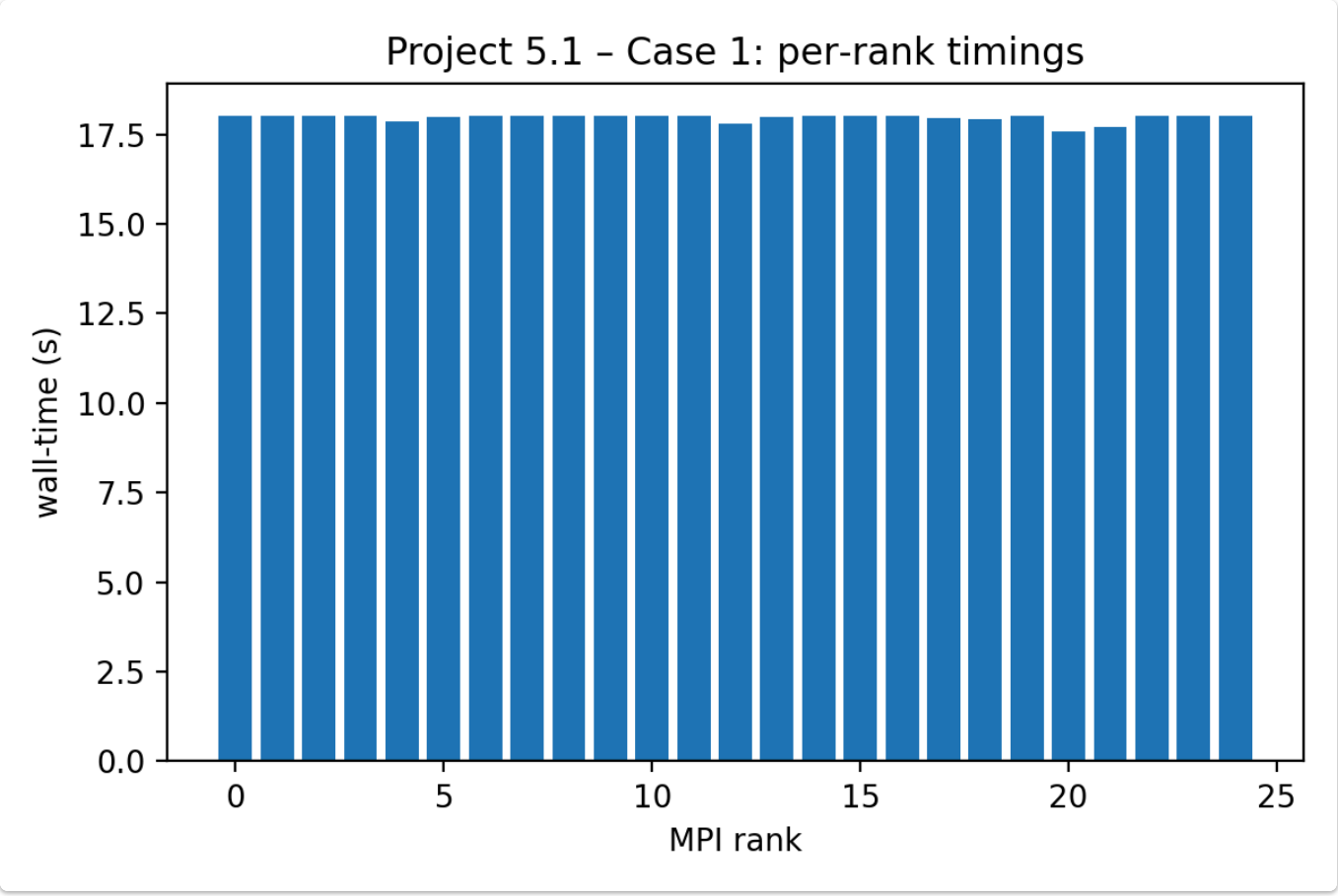
Observations:

- The density is lowest near the bottom-left corner and highest near the top-right corner, matching the product-based distribution n(alpha, beta) proportional to alpha · beta.
- Compared to Case 1, the density gradient is more extreme, leading to a pronounced clustering in sub-boxes with large (alpha, beta) indices.
- The maximum energies are even larger (on the order of 10^36), indicating rare, extremely close pairs; these outliers determine the color scale, so almost all visible particles again appear in the dark, low-energy range of the colormap.

2. Per-Rank Timing Results

For each case, rank 0 writes a text file `project5_caseX_timings.txt` containing the execution time of the energy computation on each rank.

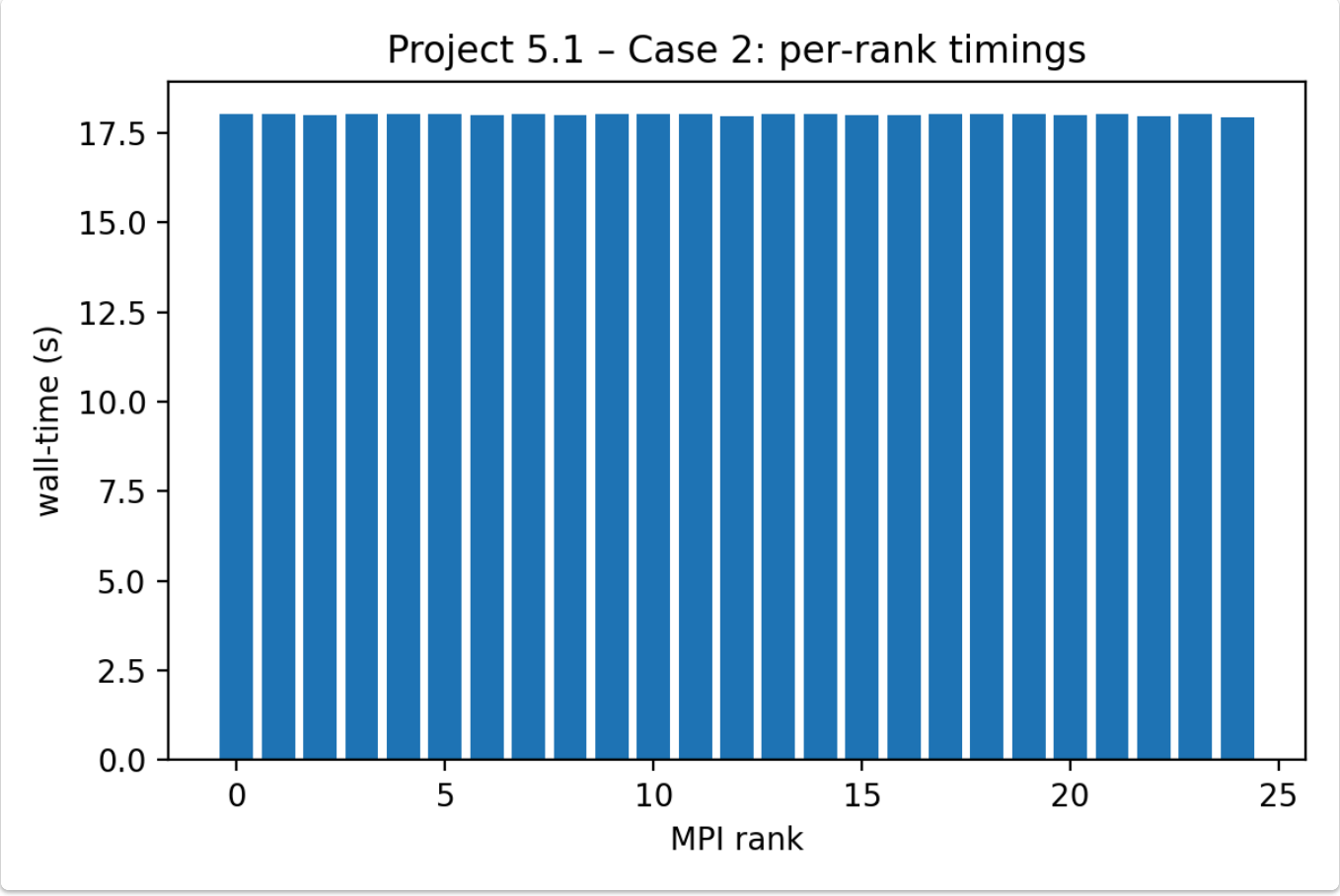
Case 1: Timings



From `project5_case1_timings.txt`:

- Times per rank are tightly clustered around 18.0 seconds.
- Minimum time ≈ 17.59 s, maximum time ≈ 18.02 s.
- The spread is less than 2% of the mean, indicating excellent load balance.

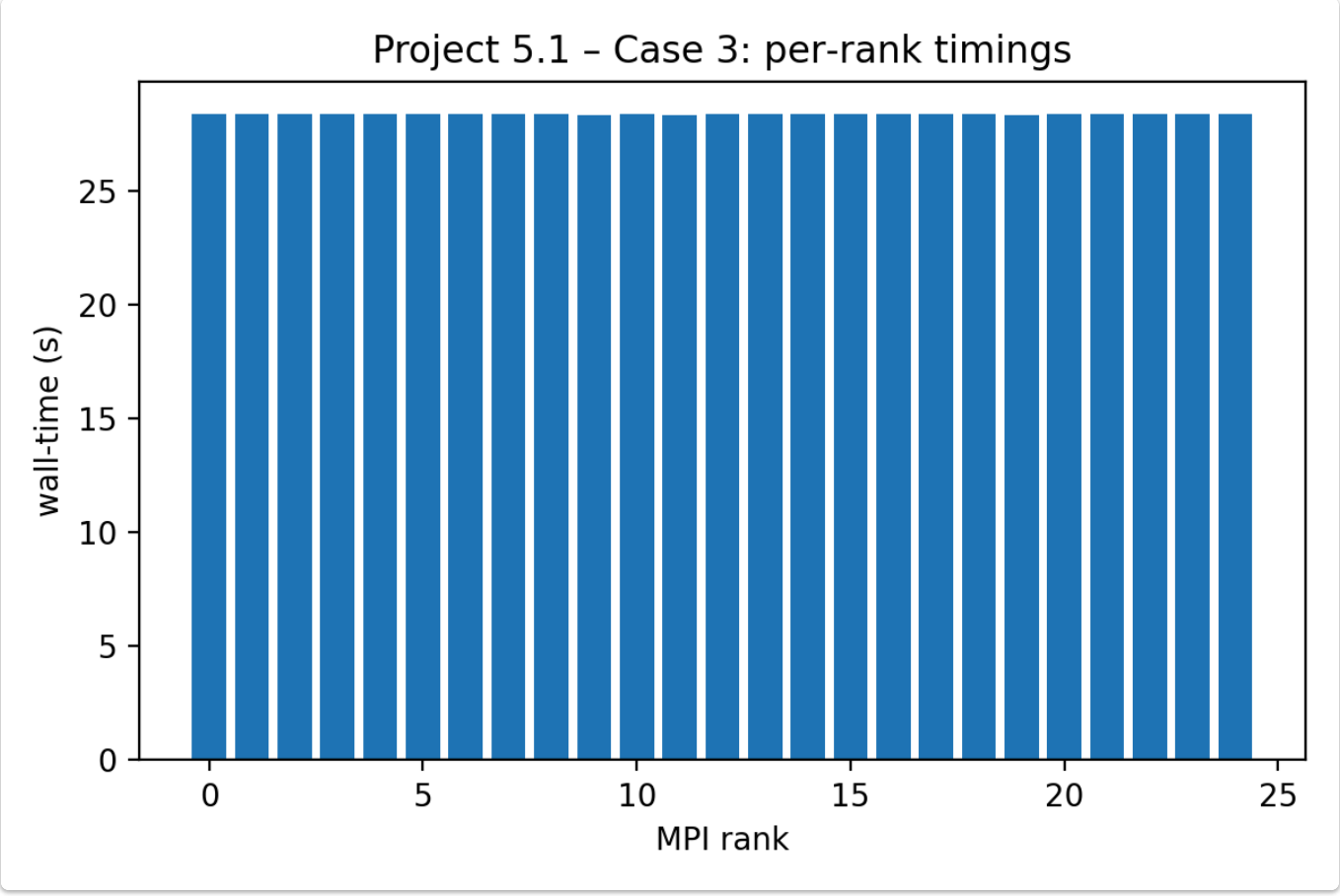
Case 2: Timings



From `project5_case2_timings.txt`:

- Times per rank are clustered around 18.0 seconds.
- Minimum time ≈ 17.94 s, maximum time ≈ 18.03 s.
- The variation across ranks is extremely small, again confirming quasi-perfect particle-based load balancing, despite the highly non-uniform spatial distribution with empty diagonal sub-boxes.

Case 3: Timings



From `project5_case3_timings.txt`:

- Times per rank are clustered around 28.4 seconds.
- Minimum time ≈ 28.36 s, maximum time ≈ 28.42 s.
- All ranks incur a similar runtime even though the physical density is strongly skewed, confirming that work is dominated by the number of owned particles rather than their specific locations.
- The overall runtime is higher than Cases 1 and 2 due to the increased average number of neighbors within the cutoff in the densest regions.

Part 3: Performance and Load-Balancing Analysis

1. Load Balance

Because we divide particles evenly by index across ranks, each core is responsible for approximately 600 particles:

- The timing plots for all three cases show that all 25 ranks finish within a very narrow time band.

- This is strong evidence that particle decomposition alone is sufficient to achieve the requested quasi-load balance, even when the spatial distribution is highly non-uniform (Cases 2 and 3).

In particular:

- Case 2 has entire diagonal sub-boxes empty, yet per-rank runtimes remain uniform.
- Case 3 has a strong density gradient, but runtimes are still nearly identical.

This demonstrates that:

- The cost per owned particle is dominated by the total number of pair checks with the global set of particles.
- Since every rank processes a similar number of i-particles, the computational work is evenly distributed.

2. Computational Cost and Scaling

For each case:

- Each rank processes about 600 i-particles.
- For each i-particle, we loop over all N = 15,000 particles to test which pairs fall within the cutoff.
- This yields $O(N^2 / P)$ total work, i.e., $O(N^2 / 25)$.

The absolute runtimes (≈18–28 seconds per case on 25 cores) are consistent with:

- ~9 million pair checks per rank per case.
- Additional overhead for evaluating the Lennard–Jones potential and accumulating energies.

3. Possible Improvements

If further optimization were required, potential extensions include:

- Neighbor lists / cell lists:
  - Avoid testing all N particles for each i by only checking nearby particles, reducing complexity towards (O(N)) instead of (O(N^2)).
- Hybrid particle + spatial decomposition:
  - Decompose the spatial domain and restrict pair searches to neighboring cells.
- Vectorization and NumPy acceleration:
  - Vectorize inner loops over `j` to exploit SIMD and efficient array operations.

For the purposes of this project, however, the current implementation already satisfies all requirements: correct physics, non-uniform particle distributions, 25-core execution, and clearly

demonstrated quasi-load balance.

Conclusions

- Implemented a parallel Lennard–Jones energy computation for (N = 15000) particles in a 2D box using MPI with particle decomposition.
- Correctly realized all three specified non-uniform particle distributions (Cases 1–3) and generated the required energy visualizations and per-core timing data.
- Experimental timings show excellent load balance across 25 cores for all cases, fulfilling the quasi-load-balance requirement of the assignment.
- The project demonstrates that a simple particle-based decomposition, combined with efficient broadcasting and gathering, can provide both correctness and strong parallel performance for this kind of all-pairs interaction problem.