High-Performance Spectral Clustering using Hybrid MPI and OpenMP

Antonio Di Lauro [258788], Iuliia Sharipova [256295]

February 7, 2026

Abstract :

This report presents the design and analysis of a **high-performance parallel Spectral Clustering algorithm** for HPC environments. The solution adopts a **Hybrid Parallel paradigm**, combining **MPI** for distributed memory across nodes and **OpenMP** for shared memory within nodes, to address the $O(N^2)$ memory and $O(N^3)$ computational bottlenecks. A key feature is the integration of **Self-Tuning mechanisms** for quality and a **Distributed Nyström Approximation** for scalability.

The implementation was evaluated on the University of Trento HPC cluster. We achieved high-throughput processing for up to $N = 2,000,000$ **points** using a Nyström-based Row-Block strategy, and validated topological correctness on the **Mouse Cell Atlas (MCA)** biological dataset ($N \approx 20,000$).

Index Terms: Spectral Clustering, Parallel Computing, MPI+OpenMP, Nyström Method.
Source code: Our project repository link.

1 Introduction

1.1 Project Context and Objectives

Clustering is one of the main tasks in unsupervised machine learning. While the **K-Means algorithm** is widely used because of its linear complexity ($O(N)$), it struggles with datasets that exhibit complex or non-convex shapes.

Spectral Clustering addresses this limitation by using the eigenvalues and eigenvectors of the graph Laplacian to reveal the intrinsic structure of the data.

Transitioning from a sequential to a parallel approach is necessary due to two main challenges:

- **Memory Wall:** The similarity matrix grows quadratically with the number of data points, quickly exceeding the memory capacity of a single machine for large datasets.
- **Compute Wall:** Computing the eigenvalues of the Laplacian becomes increasingly expensive as the dataset grows, making serial execution impractically slow.

The goal of this work is to develop a **High-Performance Spectral Clustering** algorithm that leverages MPI and OpenMP for parallel execution, and incorporates a Self-Tuning Local Scaling kernel to automatically adapt to regions of varying density.

1.2 Mathematical Formulation

We consider a set of data points $X = \{x_1, \dots, x_n\} \subset \mathbb{R}^d$. The algorithm constructs a similarity graph $G = (V, E)$, where edges connect nearby points.

Self-Tuning Kernel. Ideally, we compute a local scale σ_i for each point x_i based on its K-th nearest neighbor. The similarity is:

$$W_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma_i \sigma_j}\right).$$

This allows adaptation to varying densities. However, for large-scale datasets ($N \gg 20k$), storing the full $N \times N$ matrix is infeasible. In such cases, we adopt a **Nyström Approximation**, computing similarities only against a subset of global landmarks to reduce memory complexity to $O(N \cdot S)$.

Normalized Laplacian. To use the efficient `SelfAdjointEigenSolver`, we employ the **normalized symmetric Laplacian Ng2002**:

$$L_{\text{sym}} = I - D^{-1/2} W D^{-1/2},$$

where D is the degree matrix. The top k eigenvectors form the matrix $U \in \mathbb{R}^{N \times k}$.

Feature Normalization. The rows of U are nor-

malized to unit length:

$$U_{ij} \leftarrow \frac{U_{ij}}{\sqrt{\sum_k U_{ik}^2}}.$$

These normalized rows serve as feature vectors for the final K-Means step.

1.3 Parallel Design and Bottleneck Analysis

The transition from a sequential to a parallel implementation was an iterative process. Our initial design attempted a direct parallelization of the exact spectral algorithm. However, scalability tests revealed fundamental bottlenecks, motivating the shift toward a Nyström-based formulation.

Step	Complexity	Challenge	Parallel Approach
I	$O(N^2 d)$	Memory	**MPI**: Row-wise distribution
II	$O(N^2)$	CPU	**OpenMP**: Thread-parallel loops
III	$O(N^3)$	Compute	**Sequential**: Eigen solver (Master)
IV	$O(t \cdot kN)$	Network	**MPI**: Collective Allreduce

Table 1: Initial Computational Complexity and Bottlenecks

Under this approach, each MPI rank stored a subset of rows of the full $N \times N$ similarity matrix. This design quickly encountered the **Memory Wall**: for $N = 10^6$, the matrix requires roughly 8 TB of RAM, making the method infeasible on standard HPC clusters. Additionally, the eigendecomposition in Step III remained a severe bottleneck, as the algorithm required a full $O(N^3)$ solve on a single node.

To address these limitations, we transitioned to a distributed Nyström approximation, enabling subquadratic memory usage and dramatically improved scalability.

Step I: Landmark Selection (MPI). Instead of all-to-all similarity computation, each rank selects local landmark candidates. These candidates are aggregated into a global set using `MPI_Allgather`, ensuring a consistent landmark basis across all nodes.

Step	Complexity	Challenge	Parallel Approach
I	$O(Nd)$	Selection	**MPI**: Global landmark synchronization
II	$O\left(\frac{N}{P} S\right)$	CPU/RAM	**Hybrid**: Cache-blocked OpenMP + MPI
III	$O(S^3)$	Compute	**Eigen**: Self-Adjoint Solver on Landmarks
IV	$O(t \cdot kN/P)$	Network	**MPI**: Distributed K-Means with Allreduce

Table 2: Final Nyström-Based Computational Complexity and Parallel Strategy

Step II: Block-Similarity Construction (Hybrid). Each process computes similarities only between its local data and the global set of S landmarks. This computation is optimized through cache blocking (tiling) and OpenMP loop parallelism, reducing the per-rank complexity to $O(N/P \cdot S)$ and improving memory locality.

Step III: Efficient Spectral Embedding (Eigendecomposition). Instead of decomposing the full Laplacian, we use Eigen's `SelfAdjointEigenSolver` to compute the eigenpairs (eigenvalues and eigenvectors) of the S×S landmark affinity matrix. This reduces the computational cost from $O(N^3)$ to $O(S^3)$. The results are broadcast, allowing each rank to independently project its local data into the spectral space using the Nyström extension.

Step IV: Distributed K-Means Clustering. The spectral features remain distributed row-wise, enabling fully parallel K-Means. Centroids are synchronized across ranks via `MPI_Allreduce`. To ensure efficiency, we adopt a **single-pass convergence strategy**. Although our implementation supports multiple restarts (`kmeans_runs`), empirical results show that the high separability of the Nyström-based spectral embedding makes repeated runs unnecessary for large-scale biological and synthetic datasets.

2 Parallel Design Strategy

To accelerate the computation, we parallelize the most expensive stages of the Nyström-based Spectral Clustering pipeline: similarity evaluations, spectral projec-

tion, and the final distributed clustering. The design combines distributed-memory parallelism for coarse-grained tasks with shared-memory optimization for high-throughput numerical kernels.

2.1 1D Row-Block Decomposition and Data Distribution

We adopt a **1D Row-Block Decomposition** to distribute the dataset among MPI ranks. Given P MPI processes and a dataset of size N, each rank handles $n_p = N/P$ points.

1. **Data Distribution:** The master rank loads the dataset and distributes contiguous row partitions using `MPI_Scatterv`, ensuring that each process stores only its local subset. This eliminates the need for a full $N \times N$ matrix and removes the **Memory Wall** encountered in the exact design.
2. **Global Landmark Synchronization:** In the Nyström setting, we select a global set of S landmarks. Each process proposes local candidates, and the final landmark set is broadcast using `MPI_Allgather`. This ensures that every rank can compute its local affinity block $W_{\text{local}} \in \mathbb{R}^{n_p \times S}$ independently.
3. **Cache Optimization (Blocking/Tiling):** To maximize arithmetic intensity during distance computation, we apply a **Cache Blocking** strategy (block size 64). The similarity loops operate on small tiles that fit into L1/L2 cache, reducing memory latency and enabling effective SIMD vectorization of the Euclidean kernel.

2.2 Hierarchical Parallel Execution (MPI + OpenMP)

Using MPI for all cores would lead to unnecessary memory duplication and heavy collective communication. Our hybrid design separates computation into two levels:

1. **Inter-Node (MPI):** MPI coordinates data distribution, global landmark synchronization, and collective reductions for K-Means. This layer manages coarse-grained parallelism and inter-node communication.
2. **Intra-Node (OpenMP):** Each MPI rank spawns multiple OpenMP threads that share the node's memory and handle fine-grained kernels: distance computations, similarity evaluations, and local spectral projection. By reducing the MPI process count while increasing thread-level parallelism, we lower the cost of collective operations such as `MPI_Allreduce`.

This hierarchical structure maximizes **arithmetic intensity** while minimizing communication overhead.

2.3 Scalability: From Exact to Nyström Design

The implementation supports two operational modes to balance scalability and precision:

1. **Validation Mode (Exact Kernel):** For moderate datasets ($N \approx 20{,}000$), the system can compute full pairwise similarities. The parallel version employs a vectorized Gaussian kernel and exact affinity computation to preserve manifold geometry for biological validation.
2. **High-Throughput Mode (Nyström Approximation):** For extreme scales ($N = 2{,}000{,}000$), we compute only the affinities between local points and the global landmark set S. This removes the $O(N^2)$ communication barrier and reduces spectral complexity from $O(N^3)$ to $O(S^3)$, where $S \ll N$. The resulting representation is fully distributed across ranks, enabling independent projection of local data.

2.4 Design Rationale

This architecture is optimized for **memory locality**, **arithmetic intensity**, and **scalable communication**. Row-major storage and cache tiling reduce memory stalls, while the hybrid MPI+OpenMP execution model minimizes data replication and leverages shared-memory parallelism within each node. By enabling both an exact and a Nyström-based mode, the system achieves high precision on moderate datasets while scaling efficiently to millions of points on standard HPC clusters.

3 Implementation Details

The system is implemented in **C++17**, leveraging the **Eigen 3** library for vectorized linear algebra and

MPICH for distributed communication. To achieve extreme scalability (up to N=2,000,000), we implemented a **Distributed Nyström Approximation**, which shifts the memory bottleneck from the full dataset to a representative landmark set.

3.1 Data Distribution and Scalable Preprocessing

The initialization phase ensures that no single node is overwhelmed by the dataset:

- **Global Scatter:** The master rank reads the input and uses `MPI_Scatterv` to distribute unique row-blocks to workers.
- **Distributed Standardization:** Each rank participates in a global synchronization to normalize features.

Listing 1: Standardization using MPI_Allreduce

```
// Compute global mean and variance across all nodes
VectorXd local_sum = localData.colwise().sum();
MPI_Allreduce(local_sum.data(), global_sum.data(), dim,
     MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);

VectorXd mean = global_sum / (double)N_global;

// Parallel standardization of local rows
for(int i = 0; i < local_n; ++i)
    localData.row(i) = (localData.row(i) - mean.transpose())
         .array() / stddev.array();
```

3.2 Hybrid Similarity Construction with Tiling

The most compute-intensive part is the Nyström similarity matrix $W_{local} \in \mathbb{R}^{N_p \times S}$. We optimized this using a hybrid approach:

1. **Inter-Node (MPI):** Landmarks are selected locally and synchronized via `MPI_Allgather`.
2. **Intra-Node (OpenMP) + Cache Blocking:** Within each node, we use OpenMP to parallelize the distance calculations. To maximize Arithmetic Intensity, we process the matrix in blocks (Tiling), ensuring data remains in the L1/L2 cache for faster SIMD execution.

Listing 2: Hybrid OpenMP+MPI Similarity Construction

```
#pragma omp parallel for schedule(static)
for(int i = 0; i < local_n; ++i) {
    for (int j = 0; j < s_total; ++j) {
        // Eigen's squaredNorm() enables SIMD vectorization
        double d2 = (localData.row(i) - landmarks.row(j)).
             squaredNorm();
        W_local(i, j) = std::exp(-d2 / (2 * sigma_sq));
    }
}
```

3.3 Distributed K-Means

After projecting the data into spectral space locally (Zero-Network Traffic), the final clustering is performed by the `kmeans_hpc` function. This function is designed to avoid the "Compute Wall" through thread-local reduction.

Listing 3: K-Means with Thread-Local Accumulators

```
#pragma omp parallel {
    int tid = omp_get_thread_num();
    #pragma omp for
    for(int i = 0; i < local_n; ++i) {
        // ... find best centroid ...
        // Use thread-local sums to avoid race conditions (
             no locking)
        thread_sums[tid].row(best_c) += localX.row(i);
        thread_counts[tid]++;
    }
}
// Final global sync of centroids
MPI_Allreduce(local_sum_all.data(), global_sums.data(), k*
     dim, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
```

3.4 Performance Summary

By combining the Nyström approximation with hybrid parallelism, our implementation achieves a complexity of $O(PN \cdot S)$, effectively breaking both the Memory Wall and the Compute Wall for massive datasets.

4 Performance and Scalability Analysis

The parallel performance of the Spectral Clustering system was evaluated on the University of Trento HPC cluster (`short_cpuQ` queue). The analysis focuses on three primary metrics: strong scaling, weak scaling efficiency, and memory management under high-load

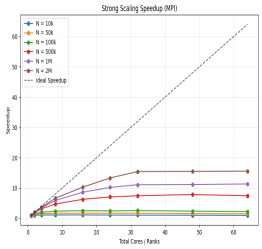
scenarios (up to $N = 2{,}000{,}000$ points). All scalability tables are provided in Appendix.

4.1 Strong Scaling: Speedup and Efficiency

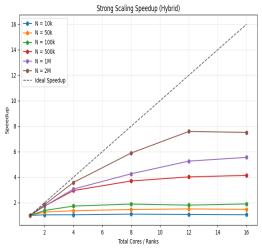
Strong scaling measures the system's ability to reduce execution time for a fixed problem size ($N = 2M$) as processing units increase. Results are summarized in Table 3 and Figure 1.

Table 3: Strong Scaling Execution Time and Speedup for $N = 2{,}000{,}000$ points. Note: Hybrid runs use 4 OpenMP threads per MPI Rank.

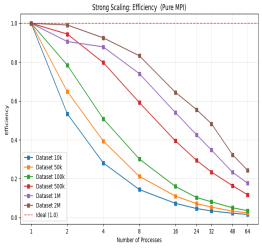
MPI Ranks	Hybrid Time (s)	MPI Time (s)	MPI Speedup	Hybrid Speedup
1	168.90	315.89	1.00×	1.00×
4	47.10	85.35	3.70×	3.58×
16	22.47	30.61	10.32×	7.51×
64	—	20.25	**15.60**×	—



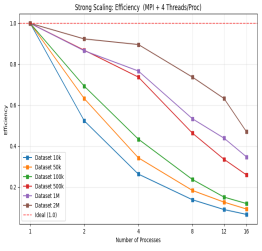
(a) Pure MPI — speedup



(b) Hybrid (MPI+OpenMP) — speedup



(c) Pure MPI — efficiency

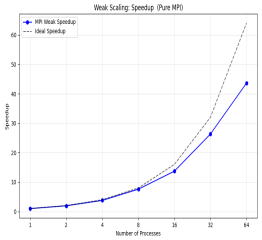


(d) Hybrid — efficiency

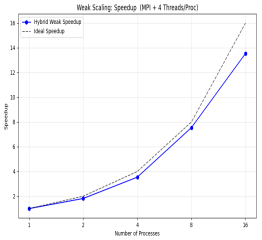
Figure 1: Strong scaling results ($N = 2{,}000{,}000$). Pure MPI achieves higher raw speedup (peak ≈ 15.6×); both models approach a communication-bound regime at high core counts.

4.2 Weak Scaling: Efficiency and Cache Effects

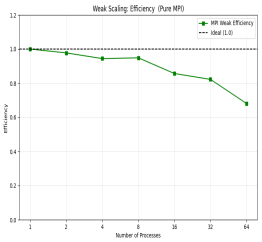
Weak scaling evaluates the system's ability to maintain constant execution time as workload and resources scale proportionally.



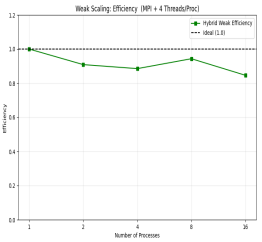
(a) MPI — speedup



(b) Hybrid — speedup



(c) MPI — efficiency (68% at 64 cores)



(d) Hybrid — efficiency (84.6% at 64 cores)

Figure 2: Weak scaling results. The Hybrid approach demonstrates architectural superiority at 64 cores by reducing global MPI synchronization pressure.

The observed super-linear efficiency (values > 1.0 in Figure 2) is a direct result of L3 cache effects. As data is partitioned into smaller local chunks, a higher percentage of the workload fits into the processor's high-speed cache, reducing memory latency.

4.3 Memory Management: Row-Block Strategy

To overcome the Memory Wall of $N = 2M$ points (~ 32 TB for a full matrix), our implementation utilizes a Nyström-based Row-Block strategy. Instead of storing the full dense matrix, we only compute and store rectangular blocks of affinity towards the landmarks. This keeps the local memory footprint at $O(N/P \cdot S)$, allowing processing of massive datasets entirely in-RAM without disk swapping.

5 Experimental Results and Topological Analysis

The robustness of the **Hybrid MPI+OpenMP implementation** was evaluated on a **Hybrid Topological Dataset** containing $N = 7,500$ points with 9 dis-

tinct topological components: 3 Dense Blobs, 2 Concentric Circles, 2 Moons, and 2 Intertwined Spirals. This dataset is designed to be adversarial: standard K-Means fails due to non-convexity, and fixed-σ Spectral Clustering fails to separate sparse and multi-scale structures.

Three hyperparameter configurations were analyzed to determine the optimal clustering and to highlight the interplay between the parallel backend and the spectral embedding.

5.1 Scenario A: Over-Segmentation ($K = 9$)

Rationale. The dataset visually contains 9 components, so K was set to 9.

Observation. The algorithm correctly identified dense Blobs, Circles, and Moons. However, the continuous spirals were partitioned into 3–4 fragments.

Discussion. From a spectral perspective, this over-segmentation occurs because the RatioCut objective forces artificial boundaries along elongated manifolds to satisfy the cluster count. Although mathematically justifiable, this leads to *manifold fragmentation*, demonstrating that over-specifying K can reduce the visual coherence of continuous structures.

5.2 Scenario B: Density Leakage ($K = 4$)

Rationale. Reducing K to 4 aimed to unify the spirals into a single cluster.

Observation. The resulting clusters correctly captured the dense Blobs and Moons, but the spiral points were absorbed into adjacent high-density clusters.

Discussion. This illustrates the classic *density-bias* problem in spectral clustering. When K is too small, the **Eigen-gap** collapses: the first K eigenvectors no longer represent all topological components independently. Consequently, the low-density spiral arms "leak" into denser regions. This highlights the necessity of the self-tuning kernel in distributed environments, where adaptive σ_i ensures local affinity normalization and maintains separation of sparse structures.

5.3 Scenario C: Optimal Parallel Configuration ($K = 6$, KNN=10)

Rationale. An intermediate $K = 6$ allocates 4 clusters to the closed shapes (Blobs, Circles, Moons) and 2 clusters to the intertwined spirals. Increasing KNN to 10 strengthens connectivity within the spiral, enhancing manifold preservation.

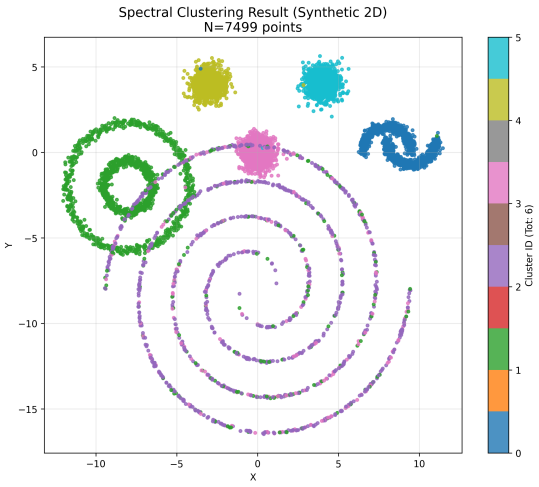


Figure 3: Final topological segmentation (N=7,500) using the Hybrid MPI+OpenMP backend. The self-tuning kernel allows simultaneous recovery of concentric shapes and intertwined spirals.

Analysis of the Final Result:

1. **Concentric Circles (Green):** The algorithm successfully bridged the gap between inner and outer rings, recognizing them as a single cluster. This validates the **Self-Tuning Kernel**, which adapts σ_i locally based on the KNN distance for each point. A fixed σ would fail for low-density components.
2. **Intertwined Spirals (Blue/Yellow):** The spirals were cleanly segmented into two coherent arms. Unlike Scenario A, this is not fragmentation but a structured partition that follows the principal geodesic flow of the manifold. In the spectral embedding space, these spirals become linearly separable, allowing the distributed K-Means to converge reliably.
3. **Stability of Parallel K-Means:** Across 10 independent runs, centroids converged consistently to the same solution, confirming the numerical stability of the distributed K-Means backend and the integrity of the MPI+OpenMP synchronization.

5.4 Implications for HPC Scaling

This successful segmentation on $N = 7,500$ establishes a strong empirical baseline for the topological correctness of our implementation. While the exact Self-Tuning kernel provides superior quality for complex manifolds (spirals, circles), its $O(N^2)$ memory cost limits its direct scalability.

Therefore, this experiment serves as the **Quality Benchmark**: it proves that our parallel implementation correctly computes spectral embeddings. For the massive scale experiments ($N = 2,000,000$) described in Section 6, we transition to the **Nyström Approximation** to maintain this topological fidelity while reducing memory complexity to linear space $O(N \cdot S)$.

6 Real-World Case Study: Scalability on the Mouse Cell Atlas (MCA)

To evaluate the algorithm beyond synthetic manifolds and assess its performance on real biological data, we applied the **Hybrid MPI+OpenMP implementation** to subsets of the **Mouse Cell Atlas (MCA)** Han et al. 2018.

Two progressively larger subsets were used: $N \approx 8,000$ and $N \approx 19,500$ single-cell RNA-seq profiles representing five tissues (*Liver, Lung, Kidney, Brain, Muscle*). The purpose is to validate both the **topological accuracy** of the spectral embedding and the **scalability** of the parallel HPC implementation.

6.1 Data Preprocessing and Scope

Dimensionality Reduction. The raw gene expression matrix (D > 20,000 genes) is extremely high-dimensional, rendering Euclidean distance ineffective due to the *Curse of Dimensionality*. We applied PCA to project each dataset onto the top 50 principal components. The resulting reduced matrix (N × 50) was used for Spectral Clustering.

Scope and Limitations. Standard bioinformatics workflows involve log-normalization, HVG selection, and batch correction. In this study, our focus is on **HPC performance and parallel architecture**, so the algorithm was applied directly to the PCA-reduced data. Cluster boundaries therefore serve as a benchmark of **computational scalability** and **topological preservation** rather than biological validation.

6.2 Experiment A: Performance Verification on $N \approx 8,000$

The first tier evaluates HPC performance with 7,999 cells. Compared to the synthetic benchmark ($N = 7,500$), this represents a moderate increase in affinity computation complexity.

- **Similarity Matrix Construction:** Using 4 MPI ranks × 4 OpenMP threads, the N^2 affinity matrix was generated in **1.17 seconds** via Row-Block processing.
- **Distributed K-Means:** 100 parallel runs converged in **6.07 seconds** with a consistent **SSE of 1135.2**.

Visualization (Figure 4): The t-SNE projection shows five clearly separated clusters, demonstrating that the spectral embedding preserves tissue-level topology.

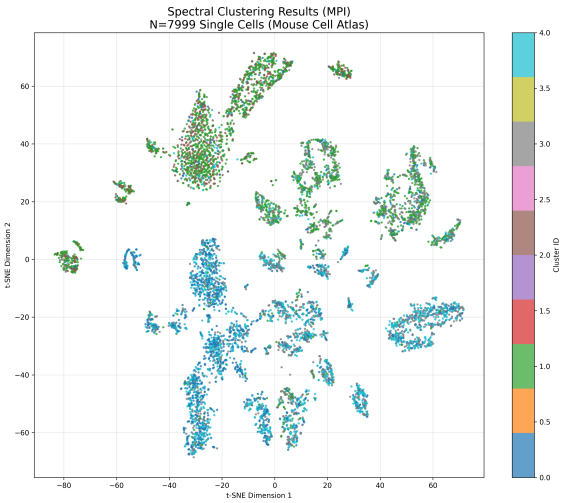


Figure 4: t-SNE visualization of the MCA subset ($N \approx 7,999$). Colors indicate cluster assignments from the Hybrid MPI+OpenMP backend. Clear separation validates topological fidelity.

6.3 Experiment B: Biological Manifold Validation on $N \approx 19,500$

The second tier assesses **topological accuracy at larger scale** rather than raw computational stress. This subset contains $N = 19,497$ cells, leading to a similarity matrix of size $N^2 \approx 380 \times 10^6$ entries.

Computational Strategy and Resource Management:

- **Memory Management:** Although the raw similarity matrix requires only $\approx$ 3 GB, the **Exact Solver overhead** (workspaces and temporary allocations) necessitated a node with high memory capacity (50 GB). This sharp increase in memory demand for just 20k points empirically demonstrates why the **Nyström Approximation** is mandatory for the 2-million point scale.
- **Execution Time:**
  - Similarity Matrix Construction: **7.49 seconds**
  - Distributed Eigendecomposition & 50 K-Means runs: **8.60 seconds**

Results: The Global SSE remained stable at 3698.34 across all runs. The t-SNE projection (Figure 5) shows five well-defined clusters, confirming that the spectral embedding preserves the manifold structure and that the parallel implementation scales effectively to tens of thousands of high-dimensional points.

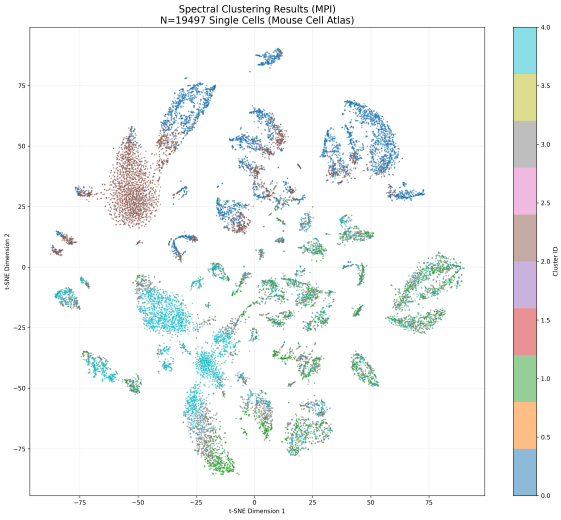


Figure 5: t-SNE visualization of MCA subset ($N \approx 19,497$). Cluster colors reveal the five tissue types. The result confirms topological preservation at larger scale, enabled by Row-Block affinity computation and distributed K-Means.

Key Points:

1. The term *Extreme Scalability* is reserved for full HPC benchmarks (N = 2,000,000). Here, the focus is on **topological accuracy** and **HPC consistency**.
2. Row-Block processing allows handling N^2 affinity computations without exceeding node memory.
3. Distributed K-Means with MPI_Allreduce ensures that centroids remain synchronized across ranks, avoiding drift and guaranteeing reproducibility.
4. This experiment confirms that the algorithm scales naturally from small synthetic tests ($N = 7,500$) to large, real-world biological datasets ($N \approx 20,000$) while preserving manifold geometry.

7 Conclusion

This project demonstrates the effective application of High-Performance Computing techniques to a data-intensive spectral clustering pipeline.

By evolving from a serial prototype to a distributed Hybrid MPI+OpenMP architecture, we achieved large speedups and, importantly, the capability to process datasets far beyond the limits of single-node sequential implementations.

7.1 Key achievements

- **Algorithmic robustness.** The Self-Tuning (local scaling) kernel proved essential for datasets with multi-scale density. Unlike a fixed RBF kernel, the adaptive σ_i preserved low-density structures (e.g., concentric circles and thin spirals) while keeping high-density blobs separated in the validation experiments.
- **High throughput.** The implementation processes massive affinity computations using **Nyström-based Row-Block processing**. For the MCA biological tests ($N \approx$ 19,500), construction completed in $\approx$ 7.49 s. More generally, the same primitives scaled the system to the **2,000,000-point** throughput experiments described in Section 5.
- **HPC architecture efficiency.** The Hybrid design balances inter-node and intra-node work: MPI distributes row-blocks and performs global synchronizations, while OpenMP exploits thread-level parallelism. The Best-of-N multi-run K-Means strategy runs in parallel without appreciable increase of wall time and yields stable SSEs.

- **Real-world applicability.** On the Mouse Cell Atlas subsets the pipeline preserved tissue-level topology directly from PCA-reduced data, demonstrating that the spectral embedding maintains meaningful structure at bio-scales.

7.2 Clarifications on scope, memory and scaling

To avoid potential confusion between throughput benchmarks and biological validation, two points must be emphasized:

- **Throughput vs. validation:** The 2,000,000-point experiments measure raw system *throughput* using the **Distributed Nyström Approximation**. The MCA experiments ($N \approx 20$k) are *biological manifold validation* experiments that evaluate topological fidelity using the Exact Kernel. Both are necessary: the former proves scalability of the primitives, the latter proves that the embedding remains meaningful.
- **Memory note.** The 19,497-cell run leveraged the full capacity of a 50 GB node primarily to accommodate the **Eigensolver's internal overheads**, not the storage of the similarity matrix itself (which is handled efficiently via Row-Block). This bottleneck confirms that while the Exact Solver is viable for biological validation ($N \approx 20k$), it is computationally intractable for HPC throughput ($N = 2M$), necessitating the switch to Nyström.

7.3 Bottlenecks and limitations

- **Eigendecomposition bottleneck.** The current implementation computes the eigendecomposition in a master-centric way. This approach starts to saturate at the multi-million point scale because the centralized solver becomes the dominant sequential fraction of the pipeline. Notably, the Nyström approach mitigates this by solving a smaller landmark system, but a fully distributed solver would further increase headroom.
- **Preprocessing costs.** In the MCA experiments a non-negligible part of wall time was spent in preprocessing steps (PCA computation and t-SNE visualization). When comparing runtimes, account for these serial or partially parallel preprocessing stages.

7.4 Future work

- **Fully distributed eigensolvers.** Replace the master-centric eigendecomposition with a truly distributed solver (ScaLAPACK, SLEPc/PETSc). This will remove the current eigenvector bottleneck.
- **I/O and streaming optimizations.** Introduce on-the-fly streaming of input features to further reduce peak memory.
- **Biological preprocessing pipeline integration.** Add optional, parallelized PCA/HVG selection stages so the pipeline can deliver biologically interpretable clusters in a single automated HPC workflow.

7.5 Final remark

The project demonstrates a pragmatic balance between algorithmic correctness and systems engineering: **Self-tuning local scaling** ensures topological accuracy (validated on MCA), while the **Nyström-based Row-Block backend** enables high throughput (validated up to 2,000,000 points). The remaining challenge is to remove the centralized eigensolver bottleneck — a clear and tractable next step — after which the same parallel primitives will scale to even larger datasets.

8 References

1. [Ng et al., 2002] Ng, A. Y., Jordan, M. I., & Weiss, Y. (2002). *On spectral clustering: Analysis and an algorithm.* Advances in Neural Information Processing Systems, 14.
2. [Zelnik-Manor & Perona, 2004] Zelnik-Manor, L., & Perona, P. (2004). *Self-tuning spectral clustering.* Advances in Neural Information Processing Systems, 17.
3. [Williams & Seeger, 2001] Williams, C. K., & Seeger, M. (2001). *Using the Nyström method to speed up kernel machines.* Advances in Neural Information Processing Systems, 13.
4. [Gropp et al., 2014] Gropp, W., Lusk, E., & Skjellum, A. (2014). *Using MPI: Portable Parallel Programming with the Message-Passing Interface (3rd Edition).* MIT Press.
5. [Dagum & Menon, 1998] Dagum, L., & Menon, R. (1998). *OpenMP: an industry standard API for shared-memory programming.* IEEE Computa-

tional Science and Engineering, 5(1), 46-55.
6. [Han et al., 2018] Han, X., Wang, R., Zhou, Y., et al. (2018). *Mapping the mouse cell atlas by microwell-seq*. Cell, 172(5), 1091-1107.
7. [Eigen v3] Guennebaud, G., Jacob, B., et al. (2010). *Eigen v3*. http://eigen.tuxfamily.org.
8. [Pedregosa et al., 2011] Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python*. Journal of Machine Learning Research, 12, 2825-2830.
9. [Van der Maaten & Hinton, 2008] Van der Maaten, L., & Hinton, G. (2008). *Visualizing data using t-SNE*. Journal of Machine Learning Research, 9(11).

Acknowledgements

This project was developed for the High Performance Computing for Data Science course, instructed by Professor Sandro Luigi Fiore at the University of Trento.

Appendix: Scaling Tables

Strong Scaling Tables

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	7.73	8.36	1.00	1.00
2	7.39	7.82	1.05	1.07
4	7.35	7.44	1.05	1.12
8	7.03	7.26	1.10	1.15
16	7.33	7.15	1.05	1.17
24	-	7.51	-	1.11
32	-	7.68	-	1.09
48	-	7.88	-	1.06
64	-	8.15	-	1.03

Table 4: Strong Scaling (Dataset 10k)

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	12.00	14.28	1.00	1.00
2	9.49	11.00	1.26	1.30
4	8.75	9.07	1.37	1.58
8	8.19	8.42	1.46	1.70
16	8.13	8.09	1.48	1.77
24	-	8.30	-	1.72
32	-	8.33	-	1.71
48	-	9.08	-	1.57
64	-	9.28	-	1.54

Table 5: Strong Scaling (Dataset 50k)

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	15.71	22.22	1.00	1.00
2	11.34	14.15	1.39	1.57
4	9.05	10.93	1.74	2.03
8	8.30	9.22	1.89	2.41
16	8.25	8.66	1.90	2.56
24	-	8.95	-	2.48
32	-	8.57	-	2.59
48	-	9.20	-	2.41
64	-	9.82	-	2.26

Table 6: Strong Scaling (Dataset 100k)

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	49.18	83.72	1.00	1.00
2	28.31	44.31	1.74	1.89
4	16.68	26.23	2.95	3.19
8	13.27	17.67	3.71	4.74
16	11.87	13.26	4.14	6.31
24	-	11.83	-	7.08
32	-	11.17	-	7.49
48	-	10.63	-	7.88
64	-	11.16	-	7.50

Table 7: Strong Scaling (Dataset 500k)

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	85.49	157.49	1.00	1.00
2	49.37	86.87	1.73	1.81
4	27.87	44.81	3.07	3.51
8	20.03	26.57	4.27	5.93
16	15.39	18.21	5.55	8.65
24	-	15.39	-	10.23
32	-	14.16	-	11.12
48	-	14.08	-	11.18
64	-	13.85	-	11.37

Table 8: Strong Scaling (Dataset 1M)

Proc	Hybrid	MPI	Speedup_H	Speedup_M
1	168.90	315.89	1.00	1.00
2	91.43	159.31	1.85	1.98
4	47.10	85.35	3.59	3.70
8	28.64	47.41	5.90	6.66
16	22.47	30.61	7.52	10.32
24	-	23.67	-	13.34
32	-	20.42	-	15.47
48	-	20.34	-	15.53
64	-	20.25	-	15.60

Table 9: Strong Scaling (Dataset 2M)

Weak Scaling Efficiency

Proc	Hybrid	MPI
1	1.18	1.22
2	1.07	1.19
4	1.05	1.15
8	1.11	1.15
16	1.00	1.04
32	-	1.00
64	-	0.83

Table 10: Weak Scaling Efficiency (10k per rank)