Segments: A Reusable Abstraction for Sparse Data on Multicore CPUs and GPUs

Tomáš Oberhuber, Jakub Klinkovský, Tomáš Halada, and Radek Fučík

Czech Technical University in Prague, Prague, Czech Republic

Abstract— Sparse matrices, graphs, numerical meshes, and particle methods organize data into variable-length groups, yet implementations often couple algorithms to storage and hardware-specific traversal. We present *Segments*, an abstraction separating logical groups from physical layout and execution policy. A segment maps a segment index and local index to a global storage index; it stores no domain-specific data. This boundary provides efficient storage, portable execution, and common traversal and reduction across multicore CPUs and GPUs. We summarize the design, sparse-matrix evaluation, and reuse in graph and mesh algorithms.

Index Terms— sparse data, performance portability, GPU, sparse matrices, graphs, numerical meshes

I Motivation and Contribution

Irregular applications process groups whose lengths differ widely. Libraries expose either concrete formats or low-level scheduling primitives, leaving applications to coordinate indexing, storage, and hardware-specific work assignment.

Segments place a stable boundary between algorithms and these choices. Applications express traversal or per-segment reduction; policies determine storage and thread assignment. Rather than adding another sparse format, Segments make formats and execution mappings interchangeable behind one API in the open-source Template Numerical Library (TNL) [1].

II Abstraction and Policies

A segment is a variable-length index space mapping segment s and local position i to a global index. Values, graph vertices, mesh entities, and other domain objects remain in application-owned arrays. The layer is therefore reusable, and one layout can index multiple fields.

The API traverses or reduces all segments, a range, a frontier, or a predicate-selected subset. Compile-time choices include CSR and ELLPACK-family layouts, permutation, and memory ordering. Run-time mappings control threads per segment and fixed or adaptive scheduling. Layout and work assignment can change without rewriting the operation.

Sparse-matrix specializations cover general matrices with explicit values, binary matrices with implicit values, and symmetric matrices storing one triangle. They retain the common interface while avoiding unnecessary storage and operations.

III Sparse-Matrix Evaluation

We evaluated sparse matrix-vector multiplication on more than 5,000 SuiteSparse matrices [2] using 32-bit indices, double precision, an NVIDIA A100, and CUDA 12.4. Median CUDA-event times were recorded after warm-up. Comparisons covered execution mappings, matrix representations, layouts, row ordering, and cuSPARSE.

Two trends emerge. Execution mapping is a first-order decision: no thread assignment is best for every segment-length distribution, while adaptive mappings cover more irregular matrices. Direct binary semantics eliminate value traffic and can broadly outperform explicit-value storage. Alternative layouts yield substantial but more specialized improvements. Global row sorting is largely neutral: 96–99% of matrices remain within 10% of the original ordering, depending on configuration. These results favor automatic selection over one fixed format and kernel.

IV Reuse Beyond Sparse Matrices

Graphs interpret segments as adjacency lists and process all vertices or a frontier without materializing a new graph. Traversal and reduction are shared building blocks for breadth-first search, shortest paths, connected components, aggregation, and message passing; each algorithm supplies its own updates and synchronization.

Numerical meshes represent topology with sparse binary incidence and adjacency matrices while keeping geometry separate. The same layouts serve cells, faces, edges, and vertices and complement TNL's GPU-capable mesh infrastructure [3]. Segments target reusable indexing and execution mechanics, not a domain-specific container.

V Conclusion and Next Steps

Segments unify irregular-data mechanics without hiding performance choices. One API now covers CSR and ELLPACK-family storage, fixed and adaptive GPU mappings, and a large matrix corpus. Next steps are automatic selection, unified filtering and post-processing, stronger frontier benchmarks, and quantitative evaluation of cross-domain API reuse.

Acknowledgment

OpenAI ChatGPT-5.6 assisted with drafting and language editing. Anthropic Claude Sonnet 5.0 assisted with data-analysis and plotting scripts. The authors verified all text and analyses and take full responsibility.

References

[1] T. Oberhuber, J. Klinkovský, and R. Fučík, "TNL: Numerical library for modern parallel architectures," *Acta Polytechnica*, vol. 61, no. SI, pp. 122–134, 2021, doi: 10.14311/AP.2021.61.0122.

[2] T. A. Davis and Y. Hu, "The University of Florida Sparse Matrix Collection," *ACM Trans. Math. Softw.*, vol. 38, no. 1, Art. 1, pp. 1–25, 2011, doi: 10.1145/2049662.2049663.

[3] J. Klinkovský, T. Oberhuber, R. Fučík, and V. Žabka, "Configurable open-source data structure for distributed conforming unstructured homogeneous meshes with GPU support," *ACM Trans. Math. Softw.*, vol. 48, no. 3, pp. 1–30, 2022, doi: 10.1145/3536164.