CLASP: Index Compression for Finite Element SpMV

Budvin Edippuliarachchi✉
budvin.edippuliarachchi@tufts.edu
Tufts University
Medford, MA, USA

Ponnuswamy Sadayappan
saday@cs.utah.edu
University of Utah
Salt Lake City, UT, USA

Hari Sundar
hari.sundar@tufts.edu
Tufts University
Medford, MA, USA

1 Introduction

Sparse Matrix-Vector multiplication (SpMV) is the dominant computational kernel in iterative solvers for partial differential equations, consuming 50–80% of wall-clock time in Finite Element Method (FEM) codes. SpMV is bandwidth-bound, making compression beneficial on CPUs and GPUs.

Sparse storage formats are well studied. Vendors (Intel-MKL, NVIDIA-cuSPARSE) support popular formats: CSR, BSR, COO, SELL, while the literature offers CSR5, SELL-C-σ, VBR, CSX, Tile-SpMV, CB-SpMV, DASP [4–9, 13]. In all these methods, block-based index compression depends on the matrix containing sufficiently dense regions - a condition FEM matrices on arbitrary meshes do not guarantee. Notably, some of these methods achieve lower SpMV performance than conventional CSR with vendor-optimized libraries (Fig. 7). Alternative FEM-specific methods include matrix-free and element-by-element (EBE) methods. Matrix-free and EBE outperform CSR only at high polynomial order ($p \geq 3$), losing at low order due to redundant computation and memory traffic [1, 3]. Instead, we decompose the operator over element clusters. The global SpMV operator is then applied in parallel per cluster, with results accumulated into the final vector - giving **Cluster Assembled Sparse Products (CLASP)** workflow. Although domain decomposition is a familiar idea in preconditioners and assembly schemes, it has not been analyzed as a compression scheme for matrix indexing.

2 Compression Scheme

Our index compression scheme is based on element clustering. FEM mesh is decomposed into element partitions, with every partition covering a bounded number (B) of nodes (i.e. DoFs). We use a Space Filling Curve (SFC) based method for element partitioning. Each nonzero entry in the sparse matrix is assigned to a partition without duplication (Fig. 1, 2). Each partition i stores its node set V_i ($|V_i| \leq B$) and its local sparse structure $A^{(i)}$ (Fig. 2). The compression stems from encoding each $A^{(i)}$ with lower-width integers. We choose B = 256 so that indices of each $A^{(i)}$ require only 1 byte - a 4x compression over conventional CSR's `int32` column indices (8x compression for larger meshes with `int64` indexing). Adjacent partitions share nodes. We show this duplication is negligible, yielding net-positive compression overall.

We define the duplication factor $D = \frac{1}{n}\sum_i^P |V_i|$ where P is the number of partitions and n is the mesh node count. Under standard assumptions (bounded valence, shape regularity, SFC partitioning), a partition of B nodes in a d-dimensional mesh has boundary size $|\delta V_i| = O(B^{(d-1)/d})$ [2, 10], giving $D = 1 + O(B^{-1/d})$. Assuming int32 (4-bytes) for conventional CSR indexing, net index savings (bytes) by using CLASP encoding is

$$\Delta I = n\big(3\bar{r} - 2 - 6(D-1)\big) - O(n/B) \qquad (\bar{r} = \text{avg. nnz per row})$$

Compression is net-positive whenever $\bar{r} > 2/3 + 2(D-1)$ which holds for FEM assembled matrices. Fig. 3 confirms this: compression improves with $\bar{r}$, while tet meshes show slightly higher D due to higher valence. We implement the compression procedure using a parallel optimized algorithm that limits the encoding overhead to the cost of ≈10 SpMV iterations on CUDA.

3 SpMV Kernels

Efficient SpMV with CLASP encoding relies on the bandwidth savings from compression, combined with asynchronous data transfer to hide latency. On NVIDIA GPUs we provide two variants of the SpMV kernel, CSR and COO, depending on how $A^{(i)}$ is encoded. In both kernels, a thread block (size 128) is assigned to each partition. After loading V_i, we stage the input vector slice in shared memory, the only uncoalesced access. After that, the implementation depends on the encoding.

CLASP-CSR: Loads local nnzs into a shared memory buffer in batches with double-buffered asynchronous operations. For each batch, each thread performs multiply-accumulate for the nnzs corresponding to at most two of its assigned rows. After processing all the batches, results are written to output vector using `atomicAdd`.

CLASP-COO: Local nnzs are fetched in batches using vectorized loads into thread-local registers while performing the multiplication. Segmented sum is calculated using batchwise `cub::BlockScan` followed by "fixes" for each row segment. Results are written back using atomics.

We also provide an x86 CPU kernel, CLASP-CSR-CPU. Partitions are parallelized across threads using OpenMP. For the fetch-FMA pattern in each local row, CLASP-CSR-CPU uses two 16-lane FMAs to maximize throughput under full `__AVX512F__` support (falling back to `__AVX2__` otherwise), while using atomic writeback.

4 Results

We evaluate the SpMV performance on 400 unstructured meshes derived from the NYU mesh dataset [11, 12]. We consider first- and second-order hex/tet elements in single and double precision (8 combinations). We compare against 7 benchmarks on NVIDIA GPUs, including cuSPARSE. On GPU, CLASP SpMV has a consistent speedup of ≈1.4x (up to 1.7x) over cuSPARSE in three NVIDIA generations (Ampere, Hopper, Blackwell) (Fig. 8). The advantage grows on newer architectures (Blackwell), which have higher compute-to-bandwidth ratios. Relative speedup generally follows the compression strength across the combinations. CLASP CPU SpMV kernel achieves up to 2x speedup over Intel-MKL in cases where the AVX vectorization is in full effect (2nd-order elements - higher nnz per row).

SC '26, Chicago, IL
2026. ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

References

[1] Julian Andrej, Nabil Atallah, Jan-Phillip Bäcker, Jean-Sylvain Camier, Dylan Copeland, Veselin Dobrev, Yohann Dudouit, Tobias Duswald, Brendan Keith, Dohyun Kim, Tzanio Kolev, Boyan Lazarov, Ketan Mittal, Will Pazner, Socratis Petrides, Syun'ichi Shiraiwa, Mark Stowell, and Vladimir Tomov. 2024. High-performance finite elements with MFEM. *The International Journal of High Performance Computing Applications* 38, 5 (2024), 447–467. arXiv:https://doi.org/10.1177/10943420241261981 doi:10.1177/10943420241261981

[2] Michael Bader. 2012. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated. doi:10.1007/978-3-642-31046-1

[3] Jed Brown, Ahmad Abdelfattah, Valeria Barra, Natalie Beams, Jean-Sylvain Camier, Veselin Dobrev, Yohann Dudouit, Leila Ghaffari, Tzanio Kolev, David Medina, Will Pazner, Thilina Ratnayaka, Jeremy L. Thompson, and Stan Tomov. 2021. libCEED: Fast algebra for high-order element-based discretizations. *Journal of Open Source Software* 6, 63 (2021), 2945. doi:10.21105/joss.02945

[4] Xing Cong, FuKai Sun, YiFan Chen, Chenhao Xie, Yi Liu, and Depei Qian. 2025. CB-SpMV:A Data Aggregating and Balance Algorithm for for Cache-Friendly Block-Based SpMV on GPUs. In *Proceedings of the 39th ACM International Conference on Supercomputing (ICS '25)*. Association for Computing Machinery, New York, NY, USA, 149–160. doi:10.1145/3721145.3725746

[5] Xing Cong, FuKai Sun, YiFan Chen, Chenhao Xie, Yi Liu, and Depei Qian. 2025. CB-SpMV:A Data Aggregating and Balance Algorithm for for Cache-Friendly Block-Based SpMV on GPUs. In *Proceedings of the 39th ACM International Conference on Supercomputing (ICS '25)*. Association for Computing Machinery, New York, NY, USA, 149–160. doi:10.1145/3721145.3725746

[6] Kornilios Kourtis, Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris. 2011. CSX: An Extended Compression Format for SpMV on Shared Memory Systems. In *Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*. ACM, 247–256. doi:10.1145/2038037.1941587

[7] Moritz Kreutzer, Georg Hager, Gerhard Wellein, Holger Fehske, and Alan R. Bishop. 2014. A Unified Sparse Matrix Data Format for Efficient General Sparse Matrix-Vector Multiplication on Modern Processors with Wide SIMD Units. In *SIAM Journal on Scientific Computing*, Vol. 36. C401–C423. doi:10.1137/130930352

[8] Weifeng Liu and Brian Vinter. 2015. CSR5: An Efficient Storage Format for Cross-Platform Sparse Matrix-Vector Multiplication. In *Proceedings of the 29th ACM on International Conference on Supercomputing* (Newport Beach, California, USA) *(ICS '15)*. Association for Computing Machinery, New York, NY, USA, 339–350. doi:10.1145/2751205.2751209

[9] Yuechen Lu and Weifeng Liu. 2023. DASP: Specific Dense Matrix Multiply-Accumulate Units Accelerated General Sparse Matrix-Vector Multiplication. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Denver, CO, USA) *(SC '23)*. Association for Computing Machinery, New York, NY, USA, Article 73, 14 pages. doi:10.1145/3581784.3607051

[10] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. 1997. Separators for sphere-packings and nearest neighbor graphs. *J. ACM* 44, 1 (Jan. 1997), 1–29. doi:10.1145/256292.256294

[11] Teseo Schneider, Yixin Hu, Xifeng Gao, Jeremie Dumas, Denis Zorin, and Daniele Panozzo. 2019. *Dataset - Thingi10k*. MIT License. https://archive.nyu.edu/handle/2451/44304

[12] Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. 2022. A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method. *ACM Trans. Graph.* 41, 3, Article 23 (March 2022), 14 pages. doi:10.1145/3508372

[13] Richard Wilson Vuduc. 2003. *Automatic Performance Tuning of Sparse Matrix Kernels*. Ph.D. Dissertation. University of California, Berkeley. https://www.proquest.com/dissertations-theses/automatic-performance-tuning-sparse-matrix/docview/305344255/se-2