CLASP: Index Compression for Finite Element SpMV

Budvin Edippuliarachchi* • **Ponnuswamy Sadayappan† (Advisor)** • **Hari Sundar* (Advisor)**

*Tufts University †University of Utah

Tufts University — School of Engineering — Computer Science

SC26 Chicago, IL | hpc unites.

Motivation

Sparse Matrix-Vector Product (SpMV) is a dominant kernel in most of numerical solvers, including the Finite Element Method (FEM). SpMV is a heavily memory-bound kernel.

$$\text{compute/mem} = \frac{2 \times nnz}{\underbrace{(4 \text{ or } 8) \times nnz}_{\text{values}} + \underbrace{(4 \text{ or } 8) \times nnz}_{\text{indexing}} + O(n)} \ll 1 \text{ FLOP/byte}$$

Indexing information in the sparse matrix accounts for 33% – 50% memory traffic. Existing work on re-encoding sparse matrices focuses only on *general* sparse matrices by detecting dense structures.

FEM sparse matrices are assembled using element-wise dense structures,

Key questions:

- Can we decompose the assembled matrix into cluster-wise denser structures?
- How much compression can we get by using smaller integers for local indexing?
- How to efficiently implement SpMV kernels using the new encoding

CLASP Encoding

Main Idea

Decompose the global SpMV operator into cluster-wise SpMV operators, each encoded using smaller-width integers. The global SpMV operator is then implemented using parallelly executed Cluster-Assembled Sparse Products.

```
Input:  Space Filling Curve (SFC) ordered mesh elements, assembled global
sparse matrix A
Output:  All partition data including offsets, CSR/COO structures, node sets
Initialize partition offsets and global output buffers.
Mark all nnzs as unassigned.
while e_i < number of elements do
   Start a running partition from element e_i.
   Add elements (e_i++) in SFC order while |partition node set| ≤ 256.
   if adding next element would exceed 256 nodes then
      Mark current partition end as e_i
   end if
   Extract this partition's nnzs from A if they were not already assigned.
   Build local CSR or COO using local indexing.
   Append partition data to outputs
end while
return partition data.
```

Fig. 1 : Algorithm for creating a partitioned CLASP encoding from an FEM sparse matrix A. We also implement an optimized parallel version that limits the encoding overhead to the cost of ≈10 SpMV iterations on CUDA.

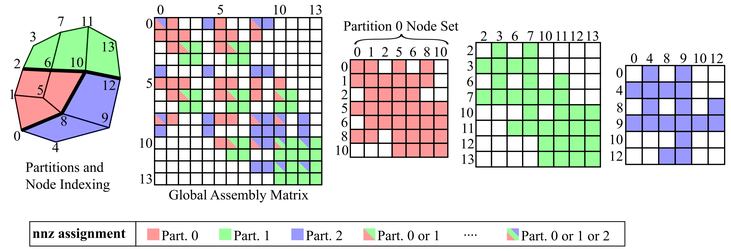


Fig. 2 : Mesh (1st-order quad) element partitions and nonzero assignment for partitions. Partitioning is defined over elements such that in each partition the covered node count ≤ B = 256. Nonzeros in the FEM matrix correspond to nodes, edges, and inner diagonals of elements (which are the FEM basis function combinations). When a nonzero touches more than one partition, it is assigned the partition with the minimum index.

Compression Bounds

Each partition is defined by the partition node set (V_i) and its local sparse structure $A^{(i)}$. V_i storage requires original indexing integer width (`int32, int64`). Indexing in $A^{(i)}$ is in a smaller width integer. Adjacent partitions contain shared nodes along their boundaries. Let P = number of partitions.

$$\text{Duplication factor (avg. node sharing) } D = \frac{1}{n}\sum_{i}^{P} |V_i|$$

	Large Partitions	Small Partitions
Duplication factor (D)	Low ▼	High ▲
$A^{(i)}$ indexing overhead	High ▲	Low ▼

For optimal local indexing we set $\max(|V_i|) = B = 256$ which allows 1 byte for local indices.

Assuming a *good* partitioning – SFC or better [2]

$$D = 1 + O(B^{-1/d}) \qquad (d = \text{dimension})$$

Index storage savings over standard CSR

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

Compression is net-positive whenever $\bar{r} > 2/3 + 2(D-1)$ which holds for FEM assembled matrices.

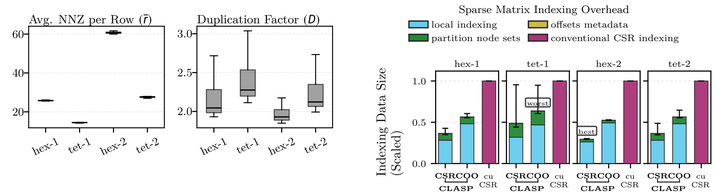


Fig. 3 : Measured values for r̄ and D from the mesh dataset (left), and observed compression for sparse indexing compared to the conventional CSR method (right). 3D hex and tet elements are used with order 1 and 2. Confirming the analysis, r̄ positively correlates with better compression rate. Higher valence in tet meshes generally increases the duplication factor D.

SpMV Kernels

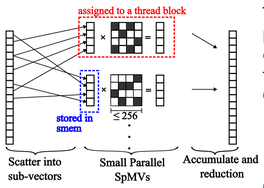


Fig. 4 : CLASP SpMV high-level view.

The decomposed SpMV operator is applied in parallel for each partition by independently loading partition-wise sparse matrices and slices from the input vector. Results are accumulated to the output vector using atomics.

On GPU,

- Thread block per partition
- CSR variant uses `memcpy_async/TMA` for *nnz* batches
- COO variant uses vectorized loads with `cub::BlockScan`

On CPU,

- OpenMP parallelization with vectorized fma, reductions

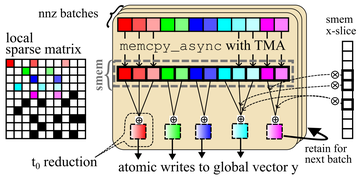


Fig. 5 : CLASP GPU SpMV - CSR variant

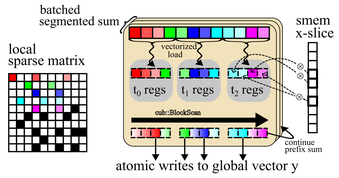


Fig. 6 : CLASP GPU SpMV - COO variant

Results

- Tested on 400 meshes (approx 10^6 elements in each) derived from the NYU mesh dataset [1] covering unstructured tet and hex meshes of both order 1 and 2.
- Compared with SOTA GPU SpMV implementations; cuSPARSE, CSR5, DASP, TileSpMV, CB-SpMV
- GPU benchmarks cover three NVIDIA GPU generations; Ampere, Hopper, Blackwell
- CPU benchmarks on Intel Sapphire Rapids with Intel-MKL

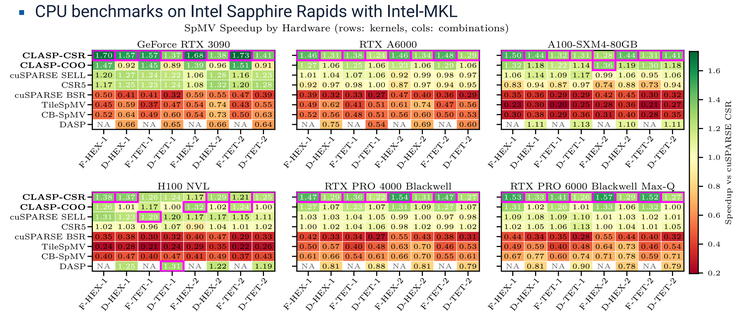


*Fig. 7 : Speedup of each SpMV implementation over cuSPARSE-SpMV. Each heatmap is for a specific GPU; rows are kernels, columns are (datatype, element shape, element order) combinations (e.g., F-HEX-1 = `float`, 1st order hexahedral elements). The best kernel in each column is highlighted in **magenta**.*

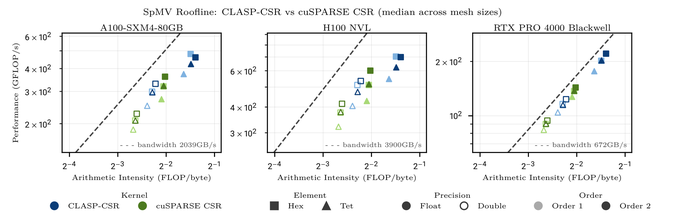


Fig. 8 : Roofline plot comparing CLASP-CSR and cuSPARSE-CSR SpMV for selected three NVIDIA GPUs.

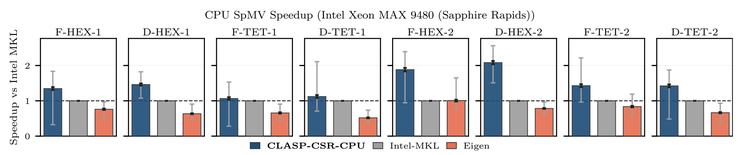


Fig. 9 : CLASP-CSR-CPU performance compared against Intel MKL and Eigen. In each plot, times are normalized to Intel MKL time. Light-colored error bars are for min-max variation, and dark-colored error bars are for 95% confidence interval.

CLASP SpMV kernels consistently outperform state-of-the-art GPU SpMV implementations across three NVIDIA generations, up to 1.7x speedup. On Intel Sapphire Rapids, CLASP CPU SpMV achieves up to 2x speedup over Intel MKL SpMV.

Summary & Future Work

- CLASP introduces provable index compression for sparse matrices generated by FEM
- SpMV with CLASP encoding consistently outperforms SOTA SpMV kernels.
- The performance advantage is more pronounced on recent architectures (e.g., Blackwell), where the ratio of computational throughput to memory bandwidth is substantially higher.
- Higher-order elements (higher *nnz* per row) achieve close to 2x speedup on CPU due to full use of AVX-512 vector instructions.
- Future work will explore cluster-wise compression of matrix values as well, enabling a fast approximate SpMV operator for use as a solver preconditioner.

ACM Student Research Competition

SC26 — The International Conference for High Performance Computing, Networking, Storage, and Analysis

Contact: Budvin.Edippuliarachchi@tufts.edu

Acknowledgements - This research was funded in part by the National Science Foundation award numbers 2513872 and 2217154.

References

[1] T. Schneider, Y. Hu, X. Gao, J. Dumas, D. Zorin, and D. Panozzo, "Dataset - thingi10k," 2019, MIT License.

[2] M. Bader, *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Publishing Company, Incorporated, 2012.

[3] Y. Niu, Z. Lu, M. Dong, Z. Jin, W. Liu and G. Tan, "TileSpMV: A Tiled Algorithm for Sparse Matrix-Vector Multiplication on GPUs," *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Portland, OR, USA, 2021, pp. 68-78.

More Info