Sparse code generation with UZP and sparse_tensor

Alonso Rodríguez-Iglesias*, Gabriel Rodríguez*, Juan Touriño*, Fabrice Rastello†

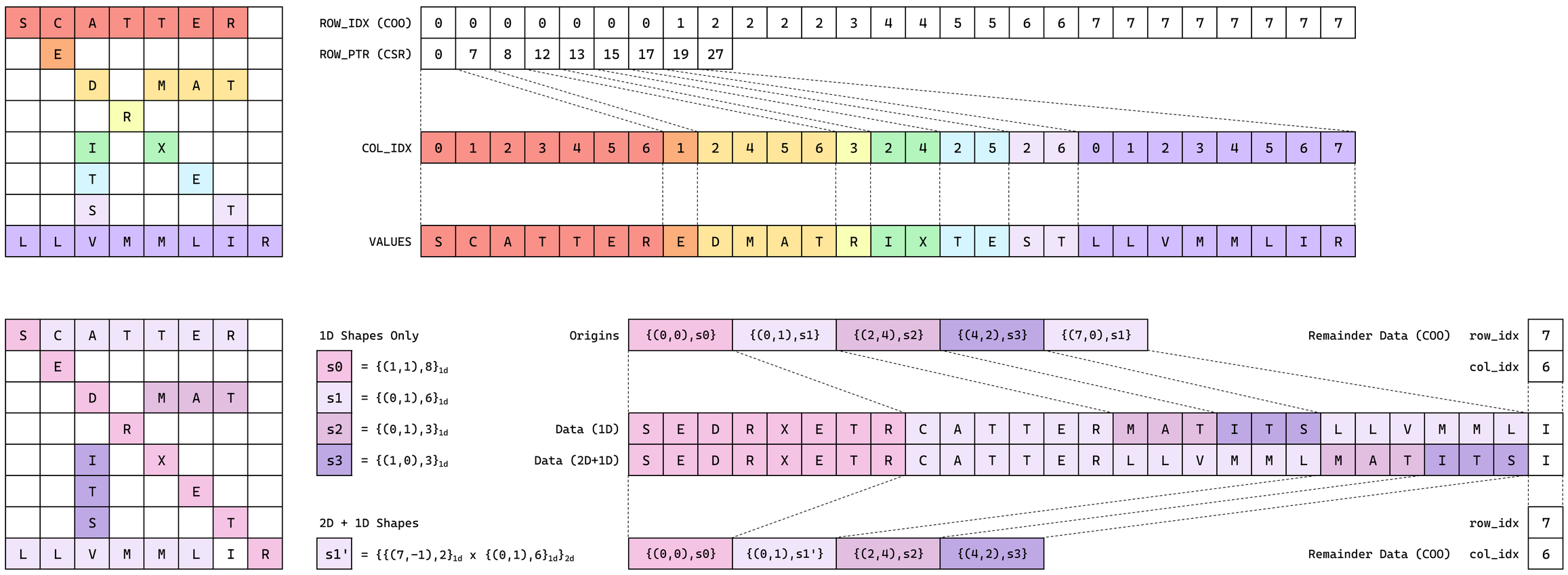
*Universidade da Coruña, GAC, CITIC, Spain. †INRIA, France. E-mail: alonso.rodriguez@udc.es

UZP Design

- A 1-d polyhedron can be understood as a 1-layer for loop, such as:

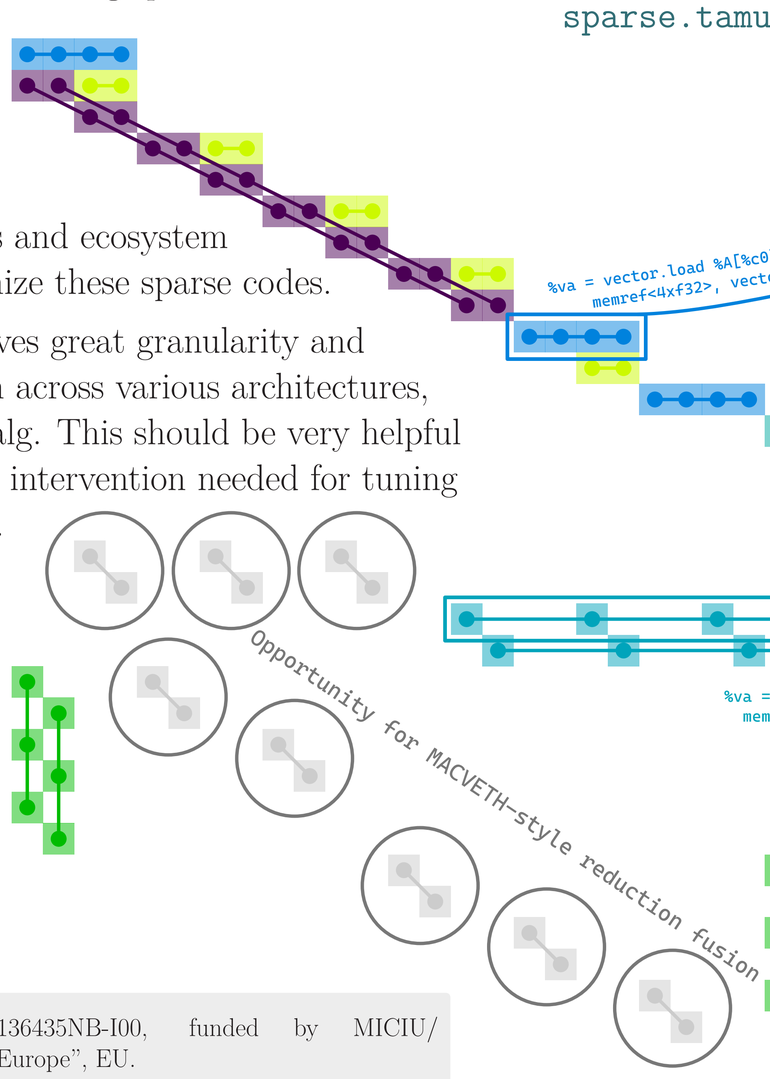
```
for(int i = 0; i < 8; i++)
    <access to> A[1+1*i][1+2*i]
```

- The loop above traverses the points (1,1), (2,3), (3,5), (4,7), (5,9), (6,11), (7,13), (8,15), and thus can be represented by a polyhedron.
- This access pattern can be represented as $\{origin, polyhedron_{md}\}$, where the m-dimensional polyhedron has *lattice* and n elements: $\{origin, \{lattice, n\}\}$, which would translate into $\{(1,1), \{(1,2), 8\}_{1d}\}$.
- A matrix can be fully represented by a collection of such polyhedra.
- When implementing sparse matrix operations, the structure of the compressed matrix itself describes the iteration domain.
- The storage system stores all encoded polyhedra sequentially, independently of its lattice.
- Based on these principles and looking for even better compression and operational efficiency, we presented the UZP (Union of $\mathcal{Z}$-Polyhedra) file format at PLDI[1], which is great at compressing, distributing and operating with matrices, created by searching for polyhedra inside sparse matrices in a way that it represents the matrix in the most compressed way possible, as well as tuning it for efficient operation.



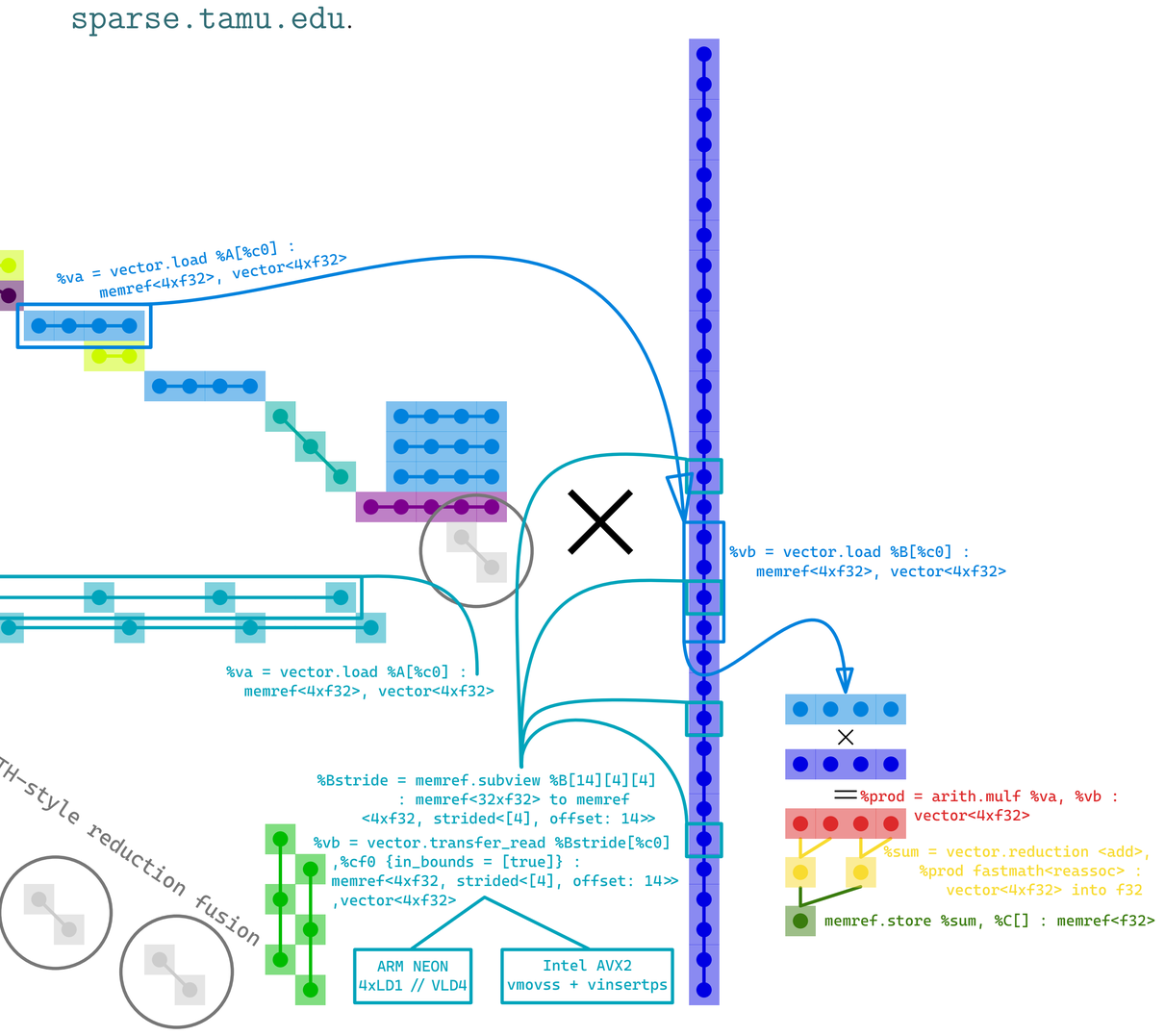
Objectives with MLIR

- To integrate UZP as a first-class storage format into the sparse / sparse_tensor MLIR dialects, as well as generating sparse code from these polyhedral structures.
- As a side effect, to make it much easier to integrate with current codebases for the end user.
- To take advantage of all the optimizations and ecosystem offered by MLIR in order to further optimize these sparse codes.
- To make UZP more portable, as MLIR gives great granularity and control over the CPU instruction set, even across various architectures, with dialects such as vector, arith and linalg. This should be very helpful in trying to eliminate the previous human intervention needed for tuning the Generic Executor (GenEx) dispatcher.



Real Pattern Visualization

- As an example, we provide a polyhedral representation view of a slightly modified (for the sake of readability) `Hamrle1` sparse matrix, from `sparse.tamu.edu`.



References

[1] A. Rodíguez-Iglesias, S. T. Tongli, E. Tucker, L.-N. Pouchet, G. Rodíguez, and J. Touriño. Modular Construction and Optimization of the UZP Sparse Format for SpMV on CPUs. *Proc. ACM Programming Langs.* 9 (PLDI), DOI: 10.1145/3729335

Acknowledgements

This work was supported by grant PID2022-136435NB-I00, funded by MICIU/AEI/10.13039/501100011033 and "ERDF A way of making Europe", EU.