Caching in on Locality: Spatial and temporal access in SpMV

Antoine Kempf[1,2,†], **Atsushi Suzuki**[3], **Masado Alexander Ishii**[3]

[1] Sorbonne Université, Paris, France
[2] Friedrich-Alexander Universität, Erlangen-Nürnberg, Germany
[3] RIKEN Center for Computational Science (R-CCS), Kobe, Japan
[†] This work was conducted during an internship program at RIKEN R-CCS

Abstract: We tackle the memory bottleneck in Sparse Matrix-Vector Multiplication (SpMV) with a memory-access-avoiding paradigm. We reduce intra-node data traffic by enhancing data locality in three ways: *spatially* with Nested Dissection, *temporally* with the Matrix Powers Kernel, and *structurally* with 4×4 Block-CSR storage. This strategy yields 2× speedup over a straightforward baseline SpMV implementation, demonstrating the importance of locality for performance on modern architectures.

Context & Motivation

Sparse Matrix-Vector Multiplication (SpMV)

- SpMV is the computational cornerstone of iterative solvers for continuum mechanics (fluid flow, heat transfer, elasticity, etc.).
- These physical problems are discretized on unstructured meshes, resulting in large sparse matrices with non-uniform patterns.

The Bottleneck

- Using standard CSR format (FP64 + 32-bits indices), the total memory traffic is high: $\sim 2.5 \times nnz + 1.5 \times nrow$.
- Since only 2 FLOPs are performed per nonzero entry, the arithmetic intensity is low: < 0.8 FLOPs per word.
- As a consequence, SpMV is strongly **memory-bound**.

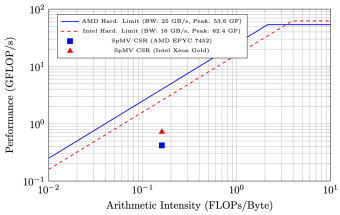

Figure 1: Roofline analysis for single-core SpMV execution on AMD EPYC 7452 and Intel Xeon Gold 5220 processors. The plot shows hardware performance limits and the achieved SpMV performance using CSR format. This figure is adapted from the first author's master's thesis.

Our Goal

While communication-avoiding algorithms [2] focus on reducing *inter*-node communication, we introduce **memory-access-avoiding** paradigm that targets the *intra*-node memory. By minimizing the Main Memory ↔ Cache traffic, we hope to lift the performance roof.

Temporal Locality

Matrix Powers Kernel (MPK)

- We target the computation of the sequence $[Ax, ..., A^k x]$, which is the heart of Krylov subspace methods.
- Interleave internal and boundary subtasks *across powers* to **immediately reuse** matrix coefficients, rather than stream through the whole matrix for each power.
- Partitioning by ND ordering allows us to compute the next vector step $A^{k+1}x$ on internal indices while the boundary indices of the current step $A^k x$ are still being processed.
- Case study: SpM2V kernel ($x \rightarrow Ax \rightarrow A^2x$).

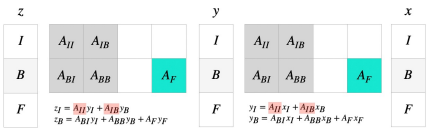

Figure 4: Local Data Flow & Dependencies ($x \rightarrow y \rightarrow z$). The highlights in Red (A_{IF}, A_{IB}) shows the blocks used for both steps: it is load once and reuse to compute z_i immediatly after y_i.

Algorithm 1: Fused SpM2V ; MPK for $\{x, y = Ax, z = A^2x\}$

```
for each subdomain p do
    // PHASE 1 : Boundary computation
    for row i ∈ [InternalEnd, End]_p do
        y[i]←A[i,:]*x[:]
    // PHASE 2 : Fused Internal computation
    for row i ∈ [Start, InternalEnd]_p do
        y[i]←A[i,:]*x[:]
    for row i ∈ [Start, InternalEnd]_p do
        z[i]←A[i,:]*y[:]
for each subdomain p do
    // PHASE 3 : Boundary computation
    for row i ∈ [InternalEnd, End]_p do
        z[i]←A[i,:]*y[:]
```

Structural Locality

Fluid Dynamics Context

- The finite element approximation for fluid flow involves 3 velocity components + 1 pressure = 4 degrees of freedom per node.
- Natural 4×4 **dense blocks** within the sparse matrix.

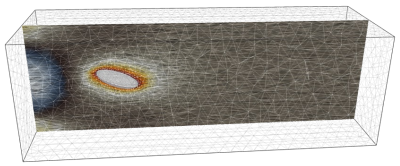

Figure 5: Visualization of 3D flow around an ellipsoid and the irregular mesh used for the simulation.

Block-CSR & SIMD Vectorization

- Block-CSR (BSR) format with block of size 4.
- The 4×4 block of FP64 values fits perfectly into 256-bit AVX2 registers with 4 vector length.
- The Fused Multiply-Add (FMA) operation is vectorized in column-major order to maximize throughput using explicit AVX2 **intrinsics** (_mm256_fmadd_pd).

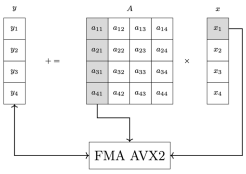

Figure 6: Schematic illustration of one fused multiply-add: the first column of A is multiplied by the corresponding entry of x and accumulated into y.

Spatial Locality

Graph Partitioning Strategy

- Since the matrices have unstructred sparsity patterns, we use graph-based algorithms to improve the data layout.
- Known methods use Reverse Cuthill—McKee (RCM) reordering to reduce the matrix bandwidth or Level-Based Blocking [1] for the computation of Matrix Powers Kernel (MPK).
- We employ METIS [3] to decompose the global matrix using **Nested Dissection** (ND) ordering.
- ND creates a hierarchy of **separator sets** and isolated subdomains, which consist of **internal** and **boundary sets**.

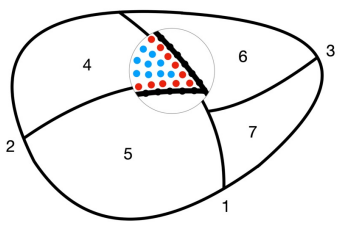

Figure 2: Geometric decomposition by Nested Dissection. The zoom illustrates the partition of subdomain 4 into Internal nodes (Blue), Boundary nodes (Red) and Separator nodes (Black).

Two-Phases Access Pattern

1) Operations are performed on the submatrix portion associated with the internal nodes. Due to reduced problem size, these submatrices hopefully **fit within the last-level cache**, accelerating vector access.
2) Off-diagonal blocks, representing connections between submatrices and separators, retain the default random-access pattern.

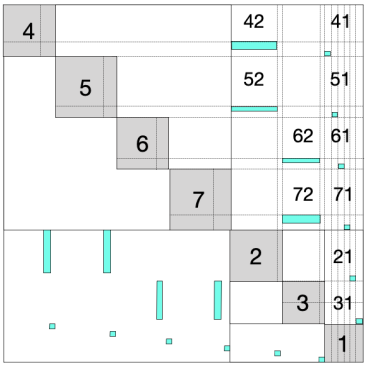

Figure 3: Global matrix reordering by Nested Dissection. The permutation isolates independant subdomains into Diagonal Blocks (Grey 4—7), pushing dependencies to the Off-Diagonal Blocks (Cyan) which connect subdomains to separators .

Results

- Hardware: AMD Ryzen 9 5950X (16-core, Zen 3) and Intel Xeon Platinium 8368 (38-core, Ice Lake).
- Test Case: GMRES solver running 100 SpMV or 50 SpM2V iterations.
- Matrix : 5-layers Nested Dissection decomposition of Stokes Matrix (n=226,820, nnz=12,984,688).
- Baseline: Standard SpMV (CSR) and Intel MKL (oneAPI 2023).

	AMD Ryzen			Intel Xeon	
Optimization step	`-mavx2`	`-mno-avx2`	MKL 2023	`-mavx2`	MKL 2023
Baseline	0.908402	4.51417	0.910328	1.74681	1.89511
+ Nested Dissection (ND)	0.810466	4.23624	0.583493	4.31193	1.79977
+ 4×4 blocking	0.486842	1.62732	0.791543	0.919048	0.925438
+ SIMD vectorization	0.470057	0.48022	—	0.788168	—
Fused SpM2V + ND	0.785550			1.36433	
+ 4×4 blocking	0.483613			0.846524	
+ SIMD vectorization	**0.440538**			**0.731669**	
Speedup	**2.06**×			**2.39**×	

- **Pre-processing overhead**: the METIS reordering takes 2.108 sec (amortized over linear/non-linear solver iterations).
- Step by step gain: ND and Register Blocking + SIMD vectorization provide the initial speedup by **improving spatial locality**.
- The fused kernel SpM2V provides the final performance leap by **exploiting temporal locality in the cache**.

Future Work

- The ND ordering naturally exposes parallelism, but the **load balance** must be treated explicitely to obtain good performance.
- Parallelization must be achieved by blocking nonzero entries of the matrix, especially on the ND separators.

References

[1] C. Alappat, G. Hager, O. Schenk, and G. Wellein, "Level-Based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 581–597, 2023
[2] J. Demmel, M. Hoemmen, M. Mohiyuddin and K. Yelick, "Avoiding communication in sparse matrix computations," *2008 IEEE International Symposium on Parallel and Distributed Processing*, Miami, FL, USA, 2008, pp. 1-12
[3] G. Karypis and V. Kumar, "A fast and high-quality multilevel scheme for partitioning irregular graphs" *SIAM Journal of Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.