

# Performance Analysis of Sparse Matrix-Vector Multiplication using OpenMP on a Multi-Socket NUMA Architecture

Davide Di Leo

Department of Information Engineering

University of Trento

Trento, Italy

davide.dileo@studenti.unitn.it

**Abstract**—Sparse Matrix-Vector multiplication (SpMV) is a memory-bound kernel widely used in scientific computing and optimization. Because of its irregular memory access patterns, performance is strongly influenced by sparsity structure, NUMA locality, and the scheduling strategy adopted by parallel frameworks such as OpenMP.

This work evaluates OpenMP performance on a 4-socket, 96-core Intel Xeon Gold system using five matrices from the SuiteSparse collection, ranging from small structured matrices to highly irregular and large-scale ones. We compare static, dynamic, and guided scheduling, analyze chunk-size effects, and study scaling behavior up to 96 threads. Cache behavior is investigated using Valgrind/Cachegrind.

Results show that static scheduling performs well on regular matrices but suffers from severe load imbalance on irregular ones. Dynamic scheduling significantly reduces imbalance at the cost of increased cache traffic. Chunk size has a measurable impact, where excessively small or large chunks degrade performance. NUMA saturation effects emerge for large matrices at high core counts.

**Index Terms**—component, formatting, style, styling, insert

## I. INTRODUCTION

Sparse Matrix-Vector Multiplication (SpMV) is a fundamental computational kernel in scientific computing, linear algebra, optimization, and machine learning. Its performance is often the dominant cost in iterative solvers such as Conjugate Gradient and GMRES. Because of its irregular memory-access behavior and low arithmetic intensity, SpMV is typically memory-bandwidth bound and difficult to scale on shared-memory architectures.

The state of the art includes numerous studies on parallel SpMV using OpenMP and NUMA architectures. Classic references such as Williams et al. “Optimizing Sparse Matrix-Vector Multiplication on Multicore Platforms” (2007) and Bell & Garland “Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors” (2009) show that scheduling strategies, memory locality, and the sparsity structure of the matrix significantly affect performance. More recent works focus on NUMA-aware partitioning and load balancing techniques for irregular matrices.

### A. Motivation

Despite extensive literature, practical performance on modern multi-socket CPUs remains highly dependent on:

- matrix structure (row length variability)
- scheduling strategy (static, dynamic, guided)
- NUMA effects
- thread oversubscription beyond a single socket.

This project aims to provide a reproducible evaluation of OpenMP scheduling policies for SpMV on a modern HPC cluster using a representative subset of matrices from the SuiteSparse Matrix Collection. The goal is to identify bottlenecks, understand scaling limits, and provide guidelines for selecting scheduling strategies.

### B. Gap

Most teaching-oriented material analyzes small matrices or synthetic benchmarks. This work fills the gap by evaluating real-world matrices of different sizes and irregularity patterns on a production HPC system.

## II. METHODOLOGY

### A. SpMV Implementation

We implement SpMV using the classical Compressed Sparse Row (CSR) format. Parallelism is applied over the rows of the matrix using an OpenMP parallel for directive. The implementation supports the following scheduling strategies:

- static
- dynamic
- guided

The parallel code computes:

$$y[i] = \sum_j A[i][j] * x[col[j]] \quad (1)$$

where  $x$  is a dense input vector generated during runtime.

TABLE I  
SELECTED MATRICES USED FOR EVALUATION.

Name	Size	Nnz	Category	Notes
1138_bus	1k × 1k	4k	Power network	Regular
bcstk25	15k × 15k	252k	Structural	Regular
G3_circuit	1.6M × 1.6M	7.6M	Circuit simulation	Irregular
nlpkt120	3.5M × 3.5M	95M	Optimization	Highly irregular
pdb1HYS	36k × 36k	4.3M	Weighted graph	Irregular

### B. Matrices

Five matrices from the SuiteSparse collection were selected to cover different sparsity patterns, sizes, and regularity levels.

The mix includes both structured problems (finite-element matrices) and unstructured graphs, enabling a comparison between load-balanced and load-imbalanced scenarios.

### C. Benchmark Platform

Experiments were run on a 4-socket Intel Xeon Gold 6252N node:

- 96 cores (4 × 24)
- 4 NUMA domains
- L3 cache per socket: 36 MB
- gcc 9.1.0 + OpenMP 4.5
- PBS job scheduler

Thread affinity:

OMP\_PLACES=cores

OMP\_PROC\_BIND=true

### D. Benchmark Procedure

- Thread counts: 1, 2, 4, 8, 12, 16, 24, 48, 96
- Schedules: static, dynamic, guided
- Chunks: 1, 4, 16
- Repeats: 10 (we report p90)

A PBS script executed all combinations automatically and produced a structured CSV file later analyzed with Python scripts.

### E. Cache Behavior via Valgrind

Because Valgrind cannot be executed in batch mode, Cachegrind experiments were performed manually in interactive sessions. Tests focused on:

- sequential execution (regular matrix)
- static vs dynamic at 24 threads
- static vs dynamic at 96 threads
- chunk=1 vs chunk=16 on irregular matrices

We collected:

- L1/L2 cache misses
- branch mispredictions
- total instructions

Results confirm the load imbalance patterns observed in runtime experiments.

## III. RESULTS

### A. Speedup

Speedup curves show clear scaling for all matrices up to 24 threads. Beyond one socket, matrices larger than cache capacity (G3\_circuit, nlpkt120) experience weaker scaling due to NUMA-induced memory bandwidth limits.

Small (regular) matrices (1138\_bus, bcstk25) exhibit almost no speedup: parallel overhead dominates the computation.

For nlpkt120 we observe that static scheduling continues to provide small gains beyond 24 threads while dynamic and guided scheduling degrade. We attribute this to NUMA and locality effects: static assigns contiguous row blocks to threads and therefore tends to keep memory accesses local to a socket, reducing remote-memory latency and preserving cache reuse. Dynamic/guided scheduling, while better for load balance, increases cross-socket memory accesses and scheduling overhead (atomic chunk allocation), producing higher cache/TLB pressure and remote memory traffic at high thread counts. We can validate this hypothesis by inspecting Cachegrind results, which show increased D1/LLd misses and branch overhead for dynamic/guided at 96 threads.

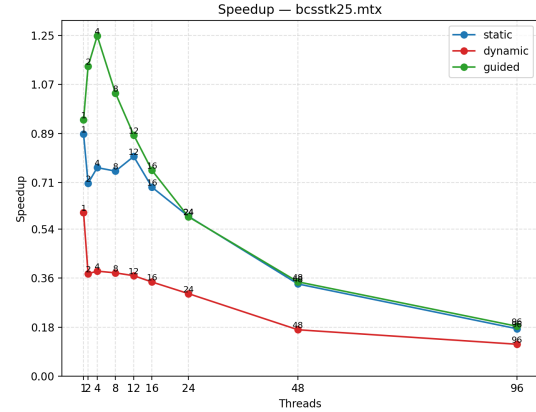


Fig. 1. Small (regular) matrices like bcstk25 exhibit almost no speedup: parallel overhead dominates the computation.

### B. Efficiency

Efficiency drops sharply beyond 24 threads for all matrices, consistent with saturating memory bandwidth per socket. Dynamic scheduling maintains slightly higher efficiency on irregular matrices.

### C. Scheduling Comparison

Static scheduling performs best on structured matrices (bcstk25), where each row has similar nonzero count. Dynamic scheduling does slightly better on irregular matrices (pdb1HYS, nlpkt120) due to load balancing. Guided scheduling behaves similarly to dynamic but shows higher overhead on irregular matrices.

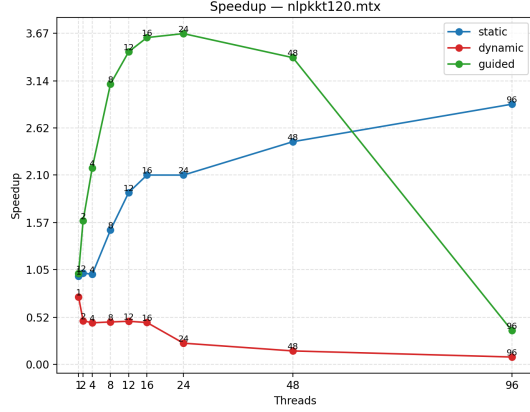


Fig. 2. nlpkkt120 speedup curves show clear scaling up to 24 threads.

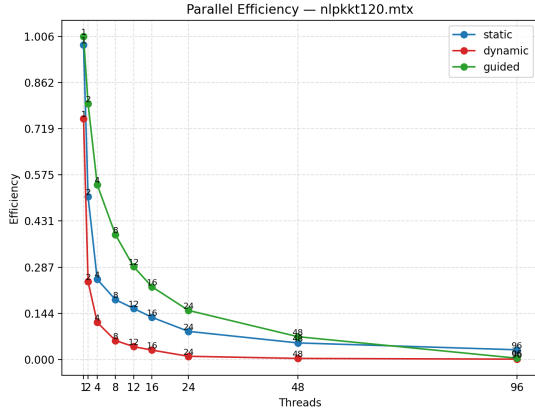


Fig. 3. nlpkkt120 efficiency curves drops shrply beyond 24 threads.

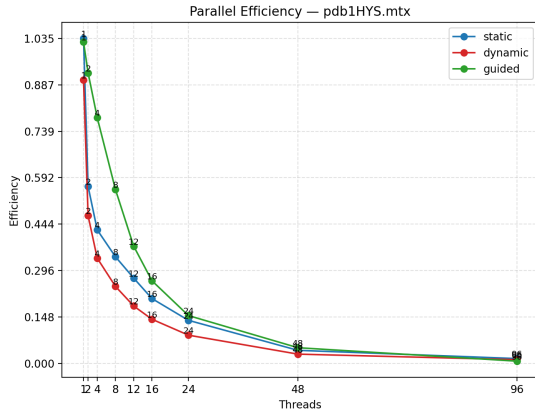


Fig. 4. Dynamic scheduling maintains slightly higher efficiency on irregular matrices like pdb1HYS.

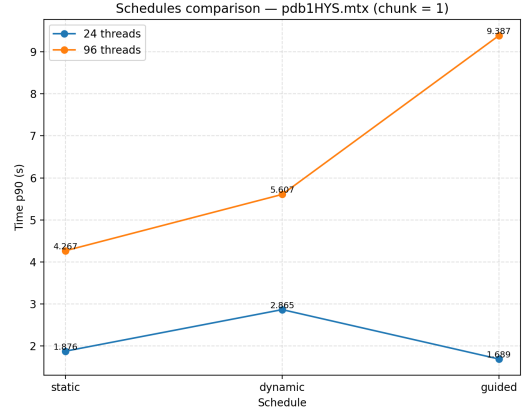


Fig. 5. On irregular matrices like pdb1HYS, dynamic scheduling shows higher overhead.

#### D. Chunk Size Effects

For irregular matrices, chunk=1 yields the best load balancing but increases scheduling overhead. Chunk=16 reduces overhead but may introduce imbalance. The difference becomes visible at 96 threads, where chunk=1 outperforms larger chunks.

#### E. Cache Behavior (Valgrind)

Cachegrind data shows:

- Static scheduling causes higher L1/L2 misses on irregular matrices due to imbalance.
- Dynamic scheduling increases instruction count (more scheduling overhead).
- Branch mispredictions are higher for dynamic, but total runtime is lower.
- LL cache miss rates remain small (1

These results validate that scheduling overhead is less important than balancing row workloads for irregular sparsity patterns.

#### F. NUMA Saturation

For thread counts above 24, the system spreads threads across sockets. This results in:

- increased memory latency (remote NUMA accesses)
- reduced scaling
- slight performance regressions at 48 and 96 threads for the largest matrices

This behavior confirms SpMV's sensitivity to memory locality.

### IV. DISCUSSION

The observed performance trends highlight the central role of memory bandwidth and sparsity structure in determining the scalability of SpMV. OpenMP scheduling choices significantly affect performance:

Static works best only when rows have similar non-zero distribution; dynamic provides robust performance across all

matrices; chunk size influences overhead vs. balance trade-offs.

NUMA architecture fundamentally limits scaling beyond one socket, with diminishing returns beyond 24 threads.

## V. CONCLUSION

This study demonstrates that OpenMP scheduling strategy is a key factor in achieving high performance in sparse matrix computations. Dynamic scheduling consistently delivers the best results on irregular matrices, while static is competitive only when row workloads are uniform.

Memory bandwidth and NUMA effects dominate scaling behavior, limiting speedup for large matrices beyond the capacity of a single socket.

The combination of runtime benchmarks and cache analysis provides a comprehensive understanding of performance bottlenecks in SpMV.

## VI. FUTURE WORK

- NUMA-aware first-touch initialization and partitioning.
- Experiments with different chunk sizes optimized per matrix.
- Comparison against vendor-tuned SpMV implementations (Intel MKL).
- Hybrid MPI+OpenMP version to test multi-node scaling, indicating bandwidth-bound behavior.
- Auto-tuning scheduling policy depending on matrix structure.

## VII. BLIOGRAPHY

- 1 S. Williams et al., “Optimization of Sparse Matrix–Vector Multiplication on Multicore Platforms,” SC’07, 2007.
- 2 N. Bell and M. Garland, “Implementing Sparse Matrix–Vector Multiplication on Throughput-Oriented Processors,” NVIDIA Technical Report, 2009.
- 3 T. A. Davis and Y. Hu, “The University of Florida Sparse Matrix Collection,” ACM Trans. Math. Softw., 2011..

## APPENDIX

The full source code, dataset, scripts, and reproducibility instructions are available at the following GitHub repository:

<https://github.com/davide-dileo/PARCO-Computing-2026-235553>.