PARCO-D2

Coppola Vincenzo
University of Trento
Trento, italy
vincenzo.coppola@studenti.unitn.it
S.I.: 237223
link to repository

Abstract—**Sparse Matrix-Vector Multiplication (SpMV) is a fundamental kernel in high-performance computing, yet its scalability on distributed memory systems is severely constrained by irregular memory access patterns and inter-process communication latency. This paper presents a comparative analysis of three parallelization strategies: a naive 1D row partitioning with vector replication, an optimized 1D split-phase execution model with communication overlapping, and a 2D Cartesian grid decomposition. These approaches have been evaluated under both strong and weak scaling regimes using a heterogeneous dataset of 8 real and 4 synthetic matrices. Results show that the naive-replicated vector approach is underwhelming for basically all of the matrices in the test set. The introduction of a split-phase strategy utilizing non-blocking MPI primitives drastically reduces the communication-to-computation ratio , achieving near-ideal weak scaling only for diagonal-ish matrices. Conversely, the 2D Cartesian decomposition proves superior in strong scaling analysis for compact dense matrices but is inherently communication bound in the long run.**

I. Introduction

SpMV is a widespread kernel in high-performance computing, essential for applications ranging from sparse neural network training to fluid dynamics. Its optimization is paramount. Yet, the irregularity of nonzero element distribution implies that no 'one-size-fits-all' strategy exists without resorting to computationally expensive preprocessing or reordering. Crucially, when scaling to distributed memory architectures, scalability is governed not by floating-point capability, but by the latency and bandwidth costs of inter-process communication. **The primary objectives of this work are to**:

- (i) **Dissect** the distributed SpMV workflow to identify critical bottlenecks and potential optimization angles.
- (ii) **Construct** distinct operational pipelines by integrating diverse parallelization strategies and communication patterns.
- (iii) **Evaluate** these implementations against a heterogeneous suite of matrices, varying significantly in sparsity structure and dimensions.
- (iv) **Analyze** the performance results using quantitative metrics, including strong and weak scaling, efficiency, and computational throughput (GFLOP/s).

II. state of the art

Boman, Devine, and Rajamanickam, in their work [BDR13], highlighted the intrinsic challenges of partitioning sparse matrices derived from scale-free graphs. Partitioning is defined as "the problem of assigning data and computation to processes. Its goal is to balance the load (data, computation) while also reducing inter-process communication". By identifying the clear inefficiency of naive 1D-based partitioning strategies, where the total communication volume grows linearly with the number of processors P due to the presence of high-degree "hub nodes" (rows packed with many nnz), the authors demonstrated the theoretical and practical superiority of 2D partitioning approaches on high imbalanced matrices. These 2D methods effectively constrain the number of neighbors each processor must communicate with to a factor of $O(\sqrt{P})$, drastically reducing network congestion and facilitating efficient load balancing across large-scale parallel systems. The work of Demmel et al. [Dem+08] focuses on minimazing communication costs by restructuring the computation to favor latency hiding. The core contribution of their work is the **PA1** algorithmic class which minimizes the communication latency by distinguishing data that is **locally computable**, meaning that they depend only on data already present on the processor allowing them to be processed in parallel while the asynchronous communication of 'outer' elements is concurrently taking place. Bienz et al. with their paper: TAP-SpMV (Topology-Aware Parallel SpMV) [BGO16] propose shaping communication patterns according to the underlying physical process topology to mitigate network contention. In this work, 'topology awareness' is achieved through a 2D Cartesian mapping implemented via MPI virtual topologies, combined with strategic hardware binding policies (–rank-by core, –bind-to core). This approach maximizes intra-node data exchange during the 'expand' phase, significantly reducing the communication volume injected into the global network fabric.

III. Methodology

A. Algorithms overview

This work compares 3 different algorithms each obtained by mixing the approaches proposed in the previously mentioned papers.

- **SPMV4**: This implementation is a naive distributed SpMV model, utilizing a 1D row-partitioning strategy with a cyclic-modulo distribution (this pattern selects contigous elements in a row which are assigned to the processess in a 'round-robin fashion' to reduce the probability of having high load imbalance).

  - **Sequential Reading**: Rank 0 is responsible for the sequential reading of the .mtx file and the subsequent distribution of matrix entries to the respective processes.
  - **Decentralized Initialization**: Each process generates a local segment of the x vector. the collective `MPI_Allgatherv` is employed so that every process replicates the full, updated x vector, allowing the subsequent calculation to proceed locally.
  - **Local Representation**: Each process builds its own local sparse representation by converting the received entries into a CSR (Compressed Sparse Row) format. This ensures that the local SpMV computation is optimized for cache efficiency before the final results are gathered.
- **1D overlapped**: This algorithm combines the split-phase execution technique that is discussed in **paper 2**, aimed at hiding the network communication latency, with the use of non blocking **point to point** MPI primitives which are used in a way that reduces the number of communication partners.
  - **Data Partitioning and Structure**: The matrix is partitioned into 1D contiguous blocks (to quantify how this strategy benefits matrix with a symmetrical diagonal structure). The overlapping of communication and computing is obtained by bifurcating the local matrix in two different CSR structures
    1) **Inner Matrix**: Contains non-zero elements whose column indices correspond to local vector elements (the diagonal block), in symbols:

       $$M_{i,j} \in \text{Inner Matrix} \iff \text{start} \leq j < \text{end} \quad (1)$$

       the local elements satisfying this property do not need to interact with the network since the correspondent elements of the x vector are already in the local memory.
    2) **Outer Matrix**: Contains "ghost" entries—non-zero elements that depend on vector segments owned by remote processes. Similarly:

       $$M_{i,j} \in \text{Outer Matrix} \iff j < \text{start} \quad \vee \quad j \geq \text{end} \quad (2)$$
  - **Asynchronous Overlapping**: The algorithm utilizes (`MPI_Isend` and `MPI_Irecv`) to initiate data transfers for remote vector elements. While these messages are in transit, the process utilizes OpenMP threads to parallelize the computation of the SpMV for the Inner Matrix.
  - **Synchronization and Completion**: After the local computation concludes, an `MPI_Waitall` barrier ensures all remote data has arrived. The process then computes the Outer Matrix contribution and aggregates it with the inner results, effectively "hiding" the network latency behind the local computational work.
- **2D overlapped**: this implemetnatation organizes the MPI processes in a (virtual) two dimensional grid (R X C) through the MPI_cart_create collective assigning coordinates to the processes. As discussed in **PAPER 2D**, this reduces the number of communication partners for each process from $P-1$ (in 1D) to approximately $\sqrt{P}$ (the processes in its same row and column).The processes are partitioned with this rule:

  $$(i,j) \in \text{Local Block} \iff i \in \text{Rows}(R) \wedge j \in \text{Cols}(C) \quad (3)$$

  The multiplication is split into two distinct communication phases, with the calculation interleaved to provide overlapping:
  1) **expand**: The processes in each grid column perform an MPI_Iallgatherv (non-blocking) to share the necessary segments of the input vector x. While this data is being transferred across the network, the CPU computes the Inner Matrix. Once the column communication completes (verified by MPI_Wait), the process computes the product for the remaining part of its local matrix
  2) **fold**: the partial results are summed across each grid row using MPI_Allreduce to produce the final elements of the output vector y

The two implementations employ different parallel I/O strategies to load the matrix data. The 1D algorithm utilizes high-performance MPI-IO collectives (MPI_File_read_at_all), allowing processes to read distinct byte chunks simultaneously and then redistribute elements to the correct owners via MPI_Alltoallv. Conversely, the 2D algorithm relies on a naive approach using standard C++ file streams (std::ifstream), where each process independently opens the file and sequentially skips through unwanted data using ignore to reach its assigned partition.

IV. Experimental setup

A. Environment & Tests

Tests were conducted on the University of Trento's HPC cluster. Each computing node was equipped with four sockets featuring four Intel(R) Xeon(R) Gold 6252N processors. Each processor operated at a base frequency of 2.30 GHz and contained 24 physical cores, totaling 96 physical cores per node. Each node was equipped with 502 GiB of RAM, providing ample capacity for large-scale sparse matrix operations. The code was compiled using the **GCC 12.3.0** suite via the mpic++ wrapper with compiler flags like O3 (high-level optimization), -march=native (architecture-specific tuning), and -std=c++17 to optimize the code parallelization. The environment utilized OpenMPI 4.1.5 for message passing and OpenMP for multithreading. The study comprises **two different distribution strategies** to evaluate the impact of data locality and network communication overhead. The first one (4 nodes/32 processes each) can count on a greater intra node shared memory communication meanwhile the second one (8 nodes/16 processes each) wants to see how the model

TABLE I: Comparative Analysis of Implementations based on Foster's Methodology

Foster Stage	SPMV4 (Baseline)	1D Overlapped	Test2D (Cartesian)
Partitioning	1D Cyclic (Modulo) distribution	1D Contiguous row-block partitioning	2D Cartesian grid ($R \times C$) partitioning
Communication	Global collective (`MPI_Allgatherv`)	Asynchronous Point-to-Point (`MPI_Isend/Irecv`)	Synchronous/Asynchronous Collectives (`Expand` & `Fold`)
Agglomeration	Local CSR conversion (Sequential)	Split-phase CSR (Inner/Outer) + OpenMP threads	2D Sub-matrix blocks + OpenMP threads
Mapping	Centralized	Hardware-aware thread binding	Virtual Topology (`MPI_Cart_create`)
Core Strategy	Load balancing for irregular matrices	Communication-Computation Overlapping	Communication volume reduction $O(N/\sqrt{P})$

behaves in setups in which it has to rely more heavily on the inter node communication. Each of these tests has been run with and without multi-threading to quantify the efficiency of the hybrid MPIxOMP . To mitigate the effects of noise and interference, tests were executed with exclusive resources allocation and including several warm up runs. Additionally the measurements were conducted with a two-level repetition protocol, inspired by the work of Hunold and Carpen-Amarie [HC16]. Their model consists of an internal measurement loop, which captures the kernel's execution time over multiple iterations, and an external macro-repetition phase, where the entire job is re-submitted five times via the PBS workload manager. For each experimental configuration (Matrix, Process Count), the final reported value is the median of the 5 "external" repetitions.

B. Tested matrices

In this work two set of matrices were considered:

- **Real matrices**: Used to do strong scaling tests. This set comprises heterogeneous matrices varying in dimension and shape, to test the kernels under different conditions. Among them can found diagonal(ish), regular/irregular patterns, tall-skinny and short-wide matrices.

TABLE II: set of real matrices

Name	Structure	Rows × Cols	nnz	Symmetry
wiki-talk-temporal	square	$1M \times 1M$	3M	21%
kron_g500-logn16	square	$65k \times 65k$	4M	100%
engine	square	$143k \times 143k$	4M	100%
t2em	square	$921k \times 921k$	4M	100%
rel9	rectangular	$9M \times 274k$	23M	0%
GL7d17	rectangular	$1.5M \times 955k$	25M	0%
GL7d21	rectangular	$822k \times 1437$	18M	0%
rail2586	rectangular	$2586 \times 274k$	8M	0%

- **Artificial matrices**: Used to do weak scaling test. This set comprises 4 different type of matrices.
  - **diag**: strict diagonal structure, best case for 0 communication
  - **rand**: pure noise, worst case that maximizes network congestion
  - **5-point-stencil**: simulates a regular 2D gridw here each point connects only to its immediate cardinal neighbors.
  - **Banded Random Matrix**: Non-zeros are confined within a bandwidth of $\sqrt{N}$ around the diagonal but are placed randomly within this band

TABLE III: Synthetic Matrices

Name	Structure	Rows (N)	nnz (approx)	Symmetry
diag{P}	Square	$100,000 \times P$	$1 \times N$	100%
rand{P}	Square	$50,000 \times P$	$20 \times N$	$\approx 0\%$
stencil{P}	Square	$10,000 \times P$	$5 \times N$	100%
banded{P}	Square	$10,000 \times P$	$20 \times N$	$\approx 0\%$

C. metrics

The recorded values are aggregated across all MPI processes using an MPI_MAX reduction to ensure that the reported time corresponds to the critical path. The reported metrics are the following:

1) MAX_IO: Time required to read the matrix from the file system and distribute the initial data structures to all processors.
2) MAX_Setup: Time spent preparing the auxiliary data structures (e.g., separating Diagonal/Off-Diagonal blocks, building the communication plan).
3) MAX_Comm: The maximum time any process spent in MPI communication routines
4) MAX_Calc: The maximum time spent by the OpenMP threads performing the actual SpMV computation
5) Total_Loop_Sec: The total wall-clock time to complete the iterative solver loop
6) Imbalance_Sec: quantifies the synchronization overhead caused by load imbalance among processors. It's computed as:

$$\text{Imbalance\_sec} = T_{\text{total\_loop\_sec}} - (\text{MAX\_Comm} + \text{MAX\_Calc})$$

7) GFLOPS: This metric measures the rate of execution independent of the problem size. Since the SpMV kernel performs exactly two floating-point operations (one multiplication and one addition) for every non-zero element (NNZ), the effective throughput is calculated as

$$\text{GFLOPS} = \frac{2 \times \text{Global\_NNZ} \times N_{iter}}{T_{loop} \times 10^9}$$

8) Speedup (strong scaling): measured as

$$S_{strong}(P) = \frac{T\_1}{T\_P}$$

9) Weak scaling:

$$E_{weak}(P) = \frac{T_{loop}^{(base\_version)}}{T_{loop}^{(P\_processes\_version)}} \times 100\%$$

10) efficiency:

$$E(P) = \frac{S\_strong(P)}{P} \times 100\%$$

V. RESULTS

The results demonstrate how memory distributed SpMV kernels are "communication-bound" applications, in particular latency bound as the number of processes increases. In general, the matrices that yielded the best scaling results are those that, trivially, could count on higher data locality, meaning that either the matrix symmetry is reasonably diagonal (e.g. engine and t2em), requiring limited designing effort, or the partitioning process successfully compacted elements that needed the same elements of the x-vector, as effect reducing the communication.

A. Considerations on Hybrid MPI + OpenMP Performance

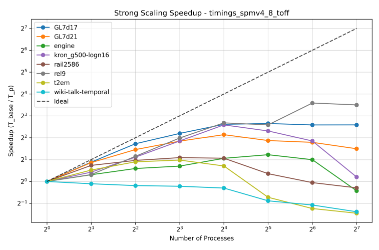
The results with multi threading on and off yielded almost equal results in terms of speedup. Since, in the results, the computation time decreases with the number of processes and tends to be much smaller than the communication time, the parallelizable fraction of the total execution time becomes insignificant, making the contribution of multi-threading negligible. To demonstrate this, follows an application of Amdahl's Law to the results of the `rail2586` matrix at $P = 128$. Let $N = 3$ be the number of OpenMP threads. The parallelizable fraction F is calculated as the ratio between the computation time (T_{calc}) and the total loop time (T_{total}), resulting in $F \approx 0.003$ (0.3%). The theoretical speedup S is therefore:

$$S = \frac{1}{(1-F) + \frac{F}{N}} = \frac{1}{(1-0.003) + \frac{0.003}{3}} \approx 1.002$$

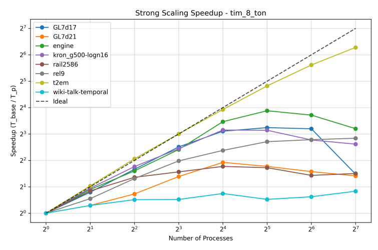
This maximum theoretical speedup of merely 0.2% confirms that the application is **strictly *communication-bound***. An hybrid parallelization targeting the computation phase cannot overcome this bottleneck unless the perfomance is pushed over the communication-bound zone.

B. Individual kernel analysis

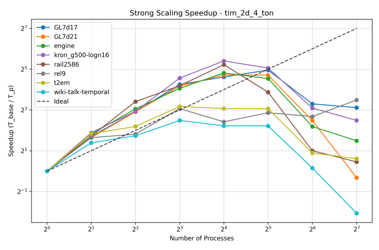
- **SPMV4**: As expected, the results immediately show limited or negative speedup for the worts cases. Replicating the entire x-vector via MPI_ALL_gatherv is just not adequate for matrices of the sizes present in the test set, making the sequential version faster than the one with 128 processes. As the number of processes increases, the aggregate GFLOPS do not scale linearly; instead, they degrade rapidly. Changing the processes mapping paradigm doesn't really affect the speedup results as this naive parallelization doesn't rely on non-blocking collectives and, rather, replicates the entire x vector locally before computing. This version gives only as much as it gets for this test set.
- **1D-Overlapped**: The adoption of a split-phase execution model combined with non-blocking MPI primitives (MPI_Isend/MPI_Irecv) has drastically altered the performance profile compared to the replicated-vector approach. Experimental results on the wiki-talk matrix show that the Communication-to-Computation Ratio has dropped from > 30 to ≈ 0.6 at 128 processes. Altough this improvements, the mapping strategy chose for this model (rows balancing) fails to balance the computational load of scale-free graphs like wiki-talk while it exceeds expectations for Banded matrices like engine and t2em. The weak scaling tests also demonstrate that the 1D Split-Phase algorithm is an optimal choice for physics-based simulations (Stencil/Banded) where locality is preserved. Its reliance on neighbor-only communication allows it to maintain near-constant runtime up to 128 processes. Ergo, a good mapping strategy is fundamental for this algorithm since the implemented split phase paradigm relies on the assumption T_{calc_inner} is roughly uniform across processes to mask T_{comm}. Despite the communication improvements, the absolute GFLOPS performance on the other real matrices remain relatively low.
- **2D-Overlapped**: The transition to a 2D MPI Cartesian topology (Test2D) has effectively solved the load balancing issue observed in 1D approaches. The Imbalance_Sec metric is consistently negligible or negative, providing empirical evidence of an adequate overlapping of computation and communication. As long as the process mapping remains intra-node (where all MPI ranks reside on the same physical node making the communication process much faster), the results yield super-linear speedup for both the 4-node and 8-node configurations. However, the performance profile changes significantly when the node capacity is exceeded and the algorithm must rely on inter-node communication (this happens respectively at 32 and 16 processes for the 4 and 8 nodes configuration). Experimental tests highlight that mapping the rows of the Cartesian communicator to contiguous physical cores yields superior performance. This mapping strategy favors the Fold phase (reduction along columns) , which becomes the critical path, whereas the cost of the Expand phase (broadcast along rows) is already partially mitigated by the split-phase computation-communication overlapping strategy. The 2D approach succeeds in two specific scenarios: Strong Locality (FEM matrices like rail/engine), where banded structures minimize off-diagonal traffic, and Hierarchical Irregularity (kron), where the 2D grid effectively diffuses the load of massive hubs that would otherwise bottleneck a 1D decomposition. In weak scaling tests, the 2D algorithm outperforms the 1D version only on random matrices. For all other matrix types, it fails because its Cartesian communication strategy, relying on collective operations in the Expand and Fold phases, introduces an



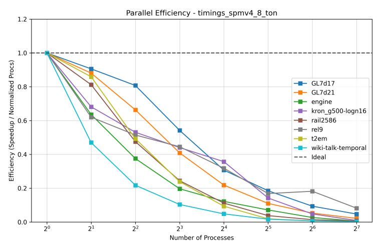
(a) SPMV4 Scaling (Threads Off)



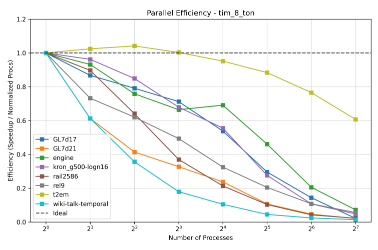
(b) Alg 1D Scaling



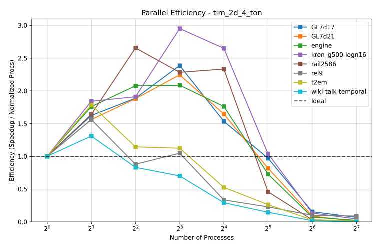
(c) Alg 2D Scaling



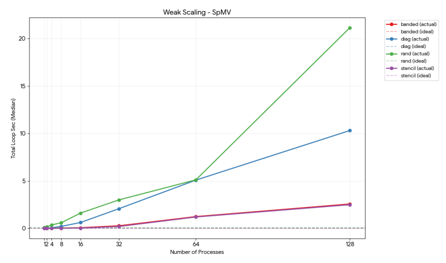
(d) SPMV4 Efficiency (Threads On)



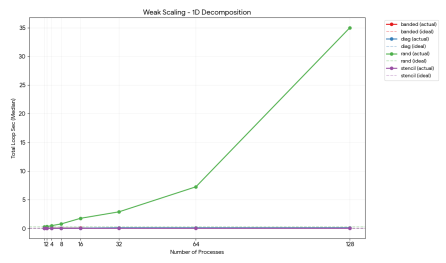
(e) Alg 1D Efficiency



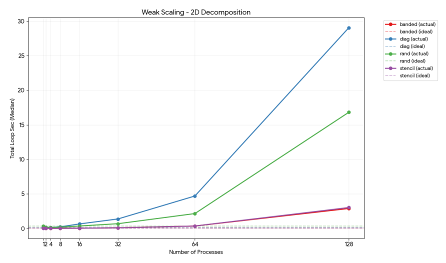
(f) Alg 2D Efficiency



(g) SPMV Weak Scaling



(h) Alg 1D Weak Scaling



(i) Alg 2D Weak Scaling

Fig. 1: Comprehensive results for 4/8-node configuration: strong scaling (top row), efficiency (middle row), and weak scaling (bottom row) across different algorithms. For reason of space, not all generated graphs are shown, but they are available in the repository.

overhead proportional to the process count, a bottleneck which is absent in the 1D approach

VI. Conclusion

The results show that there is no one-fits-all solution. The experiments proved that SpMV on distributed architectures is an operation bounded by the interconnection/communicational bottlenecks and thus, the primary directive is to optimize the communication paradigm. However, optimizing the communication protocol is insufficient without a complementary mapping and load balancing strategy, a secondary aspect that proved tantamount in relevance (the 1D approach yields near linear speedups only because the mapping strategy serves the communicational and computational aspects of the algortihm). Achieving high-performance scalability requires a conscious compromise: the solver must be structure-aware, selecting the parallelization strategy (1D vs. 2D) that best aligns the algorithmic communication pattern with the underlying topology of the matrix.

References

[BDR13] Erik G. Boman, Karen D. Devine, and Sivasankaran Rajamanickam. “Scalable matrix computations on large scale-free graphs using 2D graph partitioning”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: 10.1145/2503210.2503293. URL: https://doi.org/10.1145/2503210.2503293.

[BGO16] Amanda Bienz, William D Gropp, and Luke N Olson. “TAPSpMV: Topology-aware parallel sparse matrix vector multiplication”. In: *CoRR, vol. abs/1612.08060* (2016).

[Dem+08] James Demmel et al. “Avoiding communication in sparse matrix computations”. In: *2008 IEEE International Symposium on Parallel and Dis-*

tributed Processing. 2008, pp. 1–12. DOI: 10.1109/IPDPS.2008.4536305.

[HC16] Sascha Hunold and Alexandra Carpen-Amarie. “Reproducible MPI Benchmarking is Still Not as Easy as You Think”. In: *IEEE Transactions on Parallel and Distributed Systems* 27.12 (2016), pp. 3617–3630. DOI: 10.1109/TPDS.2016.2539167.