# Segin: Synergistically Enabling Fine-Grained Multi-Tenant and Resource Optimized SpMV

Helya Hosseini, Ubaid Bakhtiar, Donghyeon Joo, and Bahar Asgari
University of Maryland, College Park

***Abstract*—Sparse matrix-vector multiplication (SpMV) is a critical operation across numerous application domains. As a memory-bound kernel, SpMV does not require a complex compute engine but still needs efficient use of available compute units to achieve peak performance efficiently. However, sparsity causes resource underutilization. To efficiently run SpMV, we propose Segin that leverages a novel *fine-grained multi-tenancy*, allowing multiple SpMV operations to be executed simultaneously on a single hardware with minimal modifications, which in turn improves throughput. To achieve this, Segin employs hierarchical bitmaps, hence a lightweight logical circuit, to quickly and efficiently identify optimal pairs of sparse matrices to overlap. Our evaluations demonstrate that Segin can improve throughput by 1.92×, while enhancing resource utilization.**

***Index Terms*—SpMV, Multi-Tenancy, Resource Efficiency**

## I. Introduction

SPARSE matrix-vector multiplication (SpMV) is critical in fields such as scientific computing, graph analytics, and machine learning, where large, sparse matrices dominate. The high proportion of zero elements leads to challenges such as irregular memory access, limiting hardware efficiency. These challenges have driven the development of specialized accelerators and optimization techniques to improve SpMV performance [1], [4]. For example, instance, deployment of balanced tree structures [3] or HBM-based FPGA accelerators [1], [4], streamlines data retrieval and improves memory bandwidth usage.

Despite these advancements, the focus of SpMV accelerators remains on alleviating memory accesses, leaving the underutilization of hardware a significant inefficiency. In particular, in applications such as attention operations in machine learning or solvers in scientific computing, on one hand, a vector often needs to be multiplied by multiple sparse matrices; while on the other hand, as a result of sparsity, computational units frequently remain idle, leading to significant resource underutilization. To address this, we propose ***Segin***[1]*, a fine-grained multi-tenant approach that leverages idle resources to run multiple sparse workloads simultaneously, synergistically reducing inefficiencies and improving throughput*.

Segin leverages *a lightweight bitmap-based search mechanism to quickly and efficiently identify overlapping indices*. Since directly searching large matrices for optimal overlap indices is impractical, we use a hierarchical bitmap approach. In this method, elements within $chunk\_size \times chunk\_size$ blocks are mapped to a single element in a smaller bitmap matrix. This reduces the search space and accelerates the process. By efficiently identifying and pairing sparse workloads for hardware processing, this approach significantly enhances the overall efficiency of SpMV operations.

[1]Segin is a star in the northern constellation of Cassiopeia.

To implement Segin, we apply minimal changes to a targeted SpMV accelerator to select effective computations (non-zero values) and skip non-effective ones in each computation round. Additionally, the architecture tracks the results of each workload using a simple tagging mechanism. These optimization lead to more efficient utilization of computational resources. Additionally, since we can parallelize the computation of two or more matrices in a single round, throughput, defined as the number of SpMV completed per unit of time, is increased, too. In this work, we demonstrate the overlapping of two matrices, and the results indicate that throughput has increased by $1.92\times$ compared to conventional output stationary systolic arrays.

## II. Key Insights

To improve resource underutilization caused by sparsity in SpMV, our key insight is to leverage workload sparsity to enable processing multiple SpMV kernels *concurrently* in hardware rather than *sequentially*. By overlapping multiple workloads into a combined workload, which we call fine-grained multi-tenancy, the zeros of one workload are likely covered by the non-zeros of another, resulting in a denser workload. Mapping this denser workload to hardware reduces cycles spent processing zero elements, improving hardware utilization. Moreover, processing two SpMV kernels simultaneously allows completion in the same number of cycles as a single kernel in conventional hardware, boosting throughput.

Combining two or more matrices produces two scenarios: (1) non-overlapping indices, where zero elements in one matrix are covered by non-zero elements in another, creating an ideal combined matrix; and (2) overlapping non-zero elements, where some indices are shared by multiple matrices. Our approach, Segin, handles both scenarios effectively. By adding a small hardware addition into the computation unit of the systolic array, we show how Segin efficiently resolves overlapping cases. The fine-grained multi-tenancy approach, can enable the processing of two or more SpMV kernels in parallel. Increasing the number of workloads brings both opportunities and challenges. In this work, we focus on processing a pair of workloads simultaneously.

## III. Segin

Figure 1 shows a high-level overview of Segin, with steps on both the host and hardware sides. On the host side (Figure 1a), workloads are grouped into pairs of matrices. These matrices, stored in CSR format, are converted into hierarchical bitmap matrices and sent to the search unit. On the hardware side (Figure 1b), the search engine scans the matrix rows to find pairs with minimal overlaps, saving them in the pair buffer to be processed by the processing elements (PEs). Once all pairs

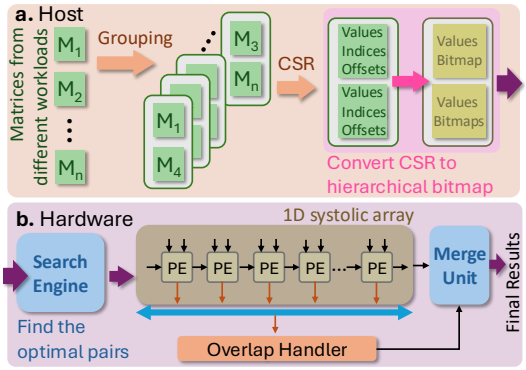


Fig. 1. High-level overview of Segin at (a) host, which includes grouping the matrices and converting CSR to hierarchical bitmap; and (b) the hardware including the search engine, the overlap handler and merge unit.

are found, the main computation begins by the PEs and the overlap handler, which deals with rare overlaps. While one set of matrices is being processed, the host sends the next group to the search unit, starting the search phase simultaneously. After computations, results from the overlap handler and PEs are merged by the merge unit, producing two vectors corresponding to the SpMV kernel of each input matrix. This process forms a pipeline with four stages: loading matrices, searching, computation, and merging results. The last three stages are collectively the computation stage.

### A. Preparing the Hierarchical Bitmaps

Bitmap compression techniques for sparse matrix representation efficiently store and manage matrices with many zero elements by leveraging sparsity patterns, thereby reducing memory and storage requirements. A bitmap, which is a 1-bit array, indicates the positions of non-zero elements and is paired with a list of these values. More efficient and flexible methods, such as hierarchical bitmaps where each bit represents a chunk of non-zero values, have been explored in the hardware and software design of prior sparse accelerators [6]. In our work, Segin utilizes hierarchical bitmapping with a granularity of $chunk\_size$, enabling quick searches and supporting multi-tenancy. Choosing an appropriate $chunk\_size$ impacts the search engine's efficiency, particularly in achieving a 100% non-overlap ratio between rows. A larger $chunk\_size$ aggregates multiple elements, potentially reducing non-overlap opportunities as it combines zero and non-zero values into a single 1-bit representation. For instance, a $chunk\_size$ of two would represent a chunk with three zero elements and one non-zero element as one, limiting the ability to exploit zero values for non-overlap. The $chunk\_size$ also affects the search engine's latency, as a smaller bitmap matrix reduces the search window. Based on our analysis (detailed in Section V), a $chunk\_size$ of 32 strikes the optimal balance between latency reduction and maximizing the non-overlap ratio.

### B. Finding the Overlaps

To find the best match for each row across matrices within a group, Segin's search algorithm operates on bitmap representations of the matrices and begins by dividing them into rows. An initial buffer is created, containing all rows from both matrices. The process starts with the first row, performing an AND operation between this row and each subsequent

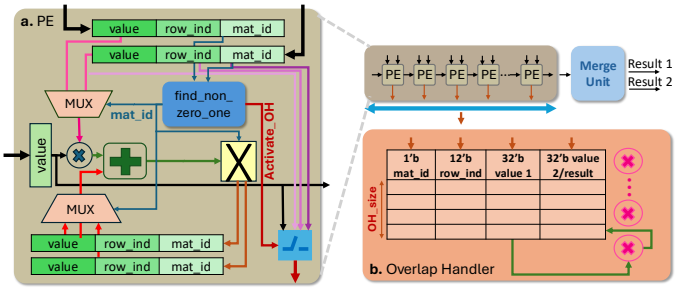


Fig. 2. Microarchitecture details of (a) PEs and (b) the Overlap Handler, within the computation stage.

row in the buffer. For every '1' resulting from the AND operation, a *num_overlaps* counter is incremented. Once all elements of the row are processed, *num_overlaps* is compared to *min_overlaps*, which tracks the fewest overlaps encountered so far. If *num_overlaps* is smaller, *min_overlaps* is updated, and the current row is recorded as the best match. The best matching pair, consisting of their matrix numbers and row indices, is added to the *final_result* buffer. The matched rows are then removed from the initial buffer. The algorithm repeats this process until all rows are paired.

### C. Mechanisms and Architecture

Segin's fine-grained multi-tenancy method is adaptable across various architectures, such as adder trees, systolic arrays, or scalar processing elements. Depending on the baseline architecture, computational units can be modified to handle multiple SpMV kernels concurrently without compromising functionality. In this paper, we target a one dimentional systolic array. Figure 2 illustrates the details of the Segin computation stage, covering modifications to the PEs, the overhead handler (OH), and the merge unit as explained below.

*1)* ***Processing Elements (PEs):*** Segin uses an output-stationary systolic array, where the vector operand streams left to right and multiplies with matrix rows streaming from the top, computing the SpMV kernel. To support multitenancy, each PE processes two inputs from the same or different matrices, using a "find_non_zero_one" module to identify nonzero elements for multiplication. The module also tags outputs with matrix IDs (mat_id) and row indices (row_ind) for partial sum updates. Overlapping workloads trigger the "activate_OH" flag, connecting the PE to the OH module.

*2)* ***Overlap Handler (OH):*** addresses rare cases where two nonzero values overlap in a PE, operating parallel to the main systolic array to avoid latency. It buffers inputs (mat_id, row_index, vector, and matrix values) and computes results when the "activate_OH" flag is triggered. To minimize memory overhead, a small, fixed-size buffer (OH_size) is used, with a search phase to update or add elements as needed.

*3)* ***Merge Unit:*** combines outputs from the OH and systolic array. It matches tags to accumulate results or appends unmatched elements from the OH. This ensures the results efficiently integrate contributions from both components.

## IV. Experimental Setup

***Simulation Infrastructure:*** We model Segin and a conventional output stationary one-dimensional systolic array (CSA) baseline using a cycle-level simulator that incorporates unit latencies from Xilinx Alveo U55C FPGA, operating at a 100

MHz clock frequency. We use the HBM on the Alveo U55C FPGA, offering 460 GB/s memory bandwidth. Additionally, to evaluate Segin's area and power, we implement its PEs and the baseline in RTL using Verilog, synthesizing them with Synopsys Design Compiler in the NANGATE 45nm library. We compare Segin's PE underutilization, throughput, area, power, and latency against the CSA and GPU implementations to assess performance and resource efficiency. For GPU comparisons, we analyze resource underutilization on an NVIDIA GTX 1650 Super using CUDA driver version 11.6 and the Nvidia Nsight toolkit. SpMV performance is evaluated with the cuSparse library using Nvidia's sample code.

***Datasets and Hardware Configurations:*** For evaluations, we use the SuiteSparse matrix collection and synthetic matrices with sparsity levels ranging from 0.1 to 0.98, ensuring diverse structural properties and a wide spectrum of sparsity. We also include some machine learning workloads that originate from the query projection matrices of LLaMA-2 7B, specifically layers 0 to 7. These workloads are pruned to a given sparsity (30%, 50%, and 70% in this paper) using Wanda's pruning algorithm [9] with no constraints on the sparsity pattern. Segin processes these datasets in $4k \times 4k$ chunks, with the NNZ column indicating the number of non-zero elements in each chunk. The performance of Segin is influenced by the following parameters:

**#PEs:** defines the systolic array size, which is set to 4K.

**OH_size:** determines the size of the buffer in OH, which depends on the number of overlapping elements we have in each timestep. Figure 3 illustrates the distribution of OH_size needed for our datasets, which indicates that the largest buffer required to store data from PEs – including vector value, matrix value, matrix_id, and row_index– is 32. Therefore, we set our OH_size to 32 to ensure sufficient capacity and provide a safety margin.

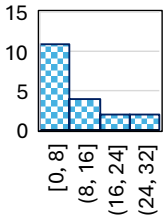

Fig. 3. Distribution of max OH_size needed in our datasets.

**Chunk_size:** Specifies the size of the hierarchical bitmap matrix (bitmap_size = mat_size/chunk_size). A chunk_size of 32 balances search engine latency and the non-overlap ratio.

## V. Evaluations

***Throughput:*** We define throughput as the number of SpMV kernels completed in one clock cycle. Since in Segin we overlap the computations of two SpMV kernels, we achieve $1.92\times$ improvement in throughput compared to processing the two SpMV kernels sequentially in a CSA. This improvement is consistent across different pairs because, in a systolic array, the number of cycles required to process an SpMV kernel is fixed and independent of the workload characteristics.

***Latency:*** Figure 4 shows the latency of two stages of the *overall pipeline* in Segin for different datasets. As is evident in this figure, the computation stage latency in Segin is similar to the loading stage. However, there are cases, such as the pair of (pf2177 - sp-0.96), where the latency of the compute stage is slightly more than the load stage. Overall, while taking advantage of Segin, we can still utilize HBM bandwidth, but the computation stage might need to stall in

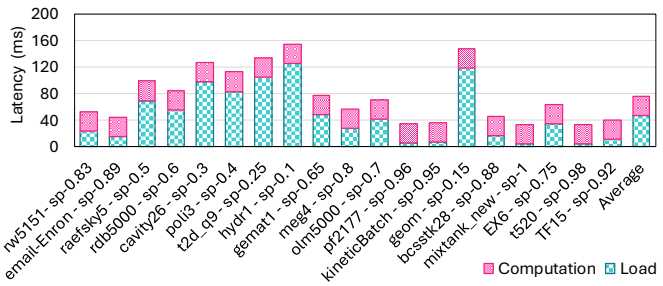

Fig. 4. Latency of pipeline stages of Segin, including the computation stage and loading from memory. "sp" stands for sparsity.

the cases where we have denser matrices in the group. In real-world applications with matrices of varying sparsity, the pipeline achieves better balance. Sparse pairs may cause idle load stages, while dense matrices make memory loading a bottleneck, stalling computation. Mixing diverse workloads balances the pipeline, with occasional stalls in either stage.

The ***computation stage*** consists of a search unit, PEs, and merge units. Using Vitis HLS, we evaluate the search engine's latency, which is influenced by the chunk_size parameter. A larger chunk_size reduces the bitmap matrix size, search window, and loop size, lowering latency. Figure 5 shows how varying the chunk size from 16 to 512 affects computation latency. We fix the chunk_size to 32. Experiments show the search unit's latency dominates the computation stage but does not hinder full utilization of HBM bandwidth.

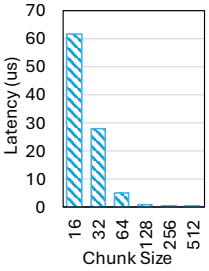

Fig. 5. Effect of chunk size on search unit latency.

In Segin, ***loading from memory*** occurs through an HBM with a bandwidth of 460 GB/s. For each pair of workloads, we need to send the bitmap matrix ($4k \times 4k$) and the actual 32-bit floating point values for sparse matrices ($NNZ \times 32$ bits). By utilizing parallel pseudo channels of HBM to transmit this data, we calculate the load stage latency based on the maximum amount of data sent in parallel. This includes two bitmap matrices of size $4k \times 4k$ bits and two values for two sparse matrices, each with a size of $32 \times$ NNZ. The larger of these determines the load stage latency.

***Resource Underutilization:*** Segin can be applied to two scenarios. First the general scenario in which the search algorithm finds the best row pairings, whether from the same matrix or different matrices based on the minimal overlap, we refer to this scenarios as the ***multi-tenancy*** approach. Second, a more specific scenario, in which the search algorithm finds the best row pairings from the same matrix, we call this scenario the ***multi-threading*** approach. Figure 6 illustrates the comparison of resource underutilization among a CSA, Segin, and GPU for the first scenario, multi-tenancy. The x-axis in this figure shows the names of workload pairs, with each pair consisting of a SuiteSparse matrix paired with a synthetic matrix by the grouping stage at the host, shown in Figure 1. The sparsity level for each synthetic matrix is indicated in the pair label; for example, "sp-0.6" means that 60% sparsity. As Figure 6 shows, Segin significantly reduces resource underutilization compared to CSA. Additionally, in many cases, Segin's PE underutilization is lower than that of

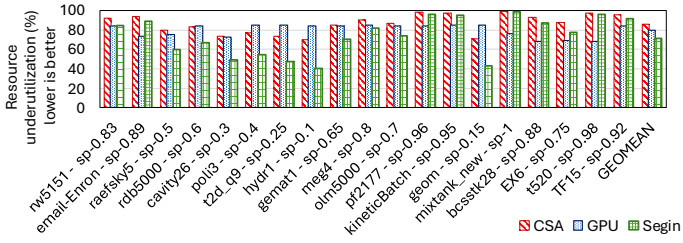


Fig. 6. Resource underutilization for multi-tenancy approach. "sp": sparsity.

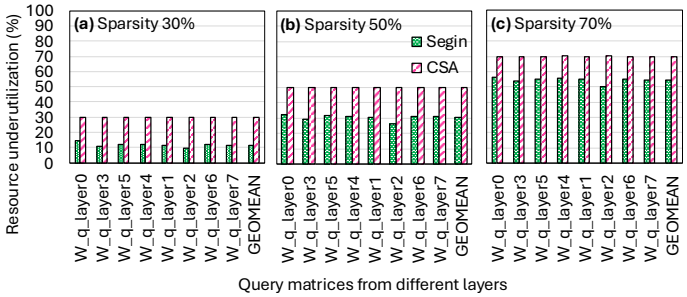


Fig. 7. Resource underutilization for multi-threading approach.

the GPU. On average, resource underutilization is improved by 14% compared to CSA and 6% compared to the GPU. This improvement results from pairing two sparse matrices with different levels of sparsity. For instance, Segin can improve resource underutilization of a pair of a matrix with a sparsity level of lower than 0.2 and SuiteSparse matrices up to 42%. However, if the random grouping process on the host side results in matrices with similar levels of sparsity in the same group, such as (t520 - sp-0.98), the improvement of Segin over CSA is not significant, and the GPU can utilize its resources more efficiently. Therefore, to leverage the benefits of Segin, we need a set of matrices on the host side with diverse sparsity. The results for the second scenario, multi-threading, shown in the Figure 7, demonstrate its effectiveness in optimizing resource utilization across targeted LLaMA-2 7B layers. Our experiments, conducted across different sparsity levels (from 30% to 70%, indicate that Segin can achieve up to a 3× improvement in resource utilization.

***Area and Power:*** Table I lists the area and power of a single PE in Segin and CSA, suggesting both the area and power of Segin have increased by factors of approximately $1.1\times$ and $1.4\times$, respectively. Therefore, we can conclude that the modifications to the PEs for Segin result in negligible power and area overhead. Also, implementing the Segin search unit in hardware results in an area of approximately $10.65\times$ that of a single Segin PE, which translates to an overall area of about $0.24\%$ of a total 4K-sized systolic array.

TABLE I
AREA AND POWER OF PES IN SEGIN AND CSA

| | Segin | CSA | Overhead |
|---|---|---|---|
| **Area ($\mu m^2$)** | 12211.185955 | 11326.555454 | 1.0781 |
| **Power (mW)** | 1.0979 | 0.7999 | 1.37254 |

## VI. Related Work

This section reviews prior efforts related to Segin, clarifying its position within the state of the art. First, systolic arrays, traditionally designed for dense computations, have been extended to support sparsity in studies such as Sparse-TPU [5], which employs an offline matrix packing that condenses sparse matrices before computation, improving PE utilization and reducing redundant MAC operations. However, Sparse-TPU relies on a preprocessing step to pack matrices before execution, making it less adaptable for dynamically changing workloads. Additionally, non-blocking simultaneous multithreading (NB-SMT) [7], [8] has been proposed to increase hardware utilization in deep learning accelerators. The SySMT framework applies NB-SMT to output-stationary systolic arrays (OS-SA), dynamically sharing resources among multiple execution flows. By temporarily reducing computation precision, SySMT maximizes hardware occupancy for deep learning workloads with minimal accuracy. Finally, while recent studies have explored executing multiple workloads on single hardware accelerators – primarily through scheduling or partitioning for multi-DNN scenarios [2], [10], [11] – their techniques are coarse-grained, which sets them apart from Segin.

## VII. Conclusions

This paper proposed Segin, a fine-grained multi-tenancy approach that reveals the benefits of overlapping sparse workloads through a hierarchical bitmap mechanism. The key insights underscored how strategic pairing of sparse matrices can lead to more efficient resource utilization, which can inspire novel directions for optimizing computation in SpMV.

## Acknowledgments

This work is supported by the NSF PPoSS program, under Award Number 2316177.

## References

[1] Y. Du, Y. Hu, Z. Zhou, and Z. Zhang. High-performance sparse linear algebra on hbm-equipped fpgas using hls: A case study on spmv. In *FPGA*, pages 54–64, 2022.

[2] H. Fan, S. Venieris, A. Kouris, and N. Lane. Sparse-dysta: Sparsity-aware dynamic and static scheduling for sparse multi-dnn workloads. In *MICRO*, pages 353–366, 2023.

[3] S. Feng, X. He, K.-Y. Chen, L. Ke, X. Zhang, D. Blaauw, T. Mudge, and R. Dreslinski. Menda: A near-memory multi-way merge solution for sparse transposition and dataflows. In *ISCA*, pages 245–258, 2022.

[4] A. Gerami and B. Asgari. Gust: Graph edge-coloring utilization for accelerating sparse matrix vector multiplication. In *ASPLOS*, 2024.

[5] X. He, S. Pal, A. Amarnath, S. Feng, D.-H. Park, A. Rovinski, H. Ye, Y. Chen, R. Dreslinski, and T. Mudge. Sparse-tpu: Adapting systolic arrays for sparse matrices. In *SC*, pages 1–12, 2020.

[6] K. Kanellopoulos, N. Vijaykumar, C. Giannoula, R. Azizi, S. Koppula, N. M. Ghiasi, T. Shahroodi, J. G. Luna, and O. Mutlu. Smash: Co-designing software compression and hardware-accelerated indexing for efficient sparse matrix operations. In *MICRO*, pages 600–614, 2019.

[7] G. Shomron, T. Horowitz, and U. Weiser. Smt-sa: Simultaneous multithreading in systolic arrays. *IEEE Computer Architecture Letters*, 18(2):99–102, 2019.

[8] G. Shomron and U. Weiser. Non-blocking simultaneous multithreading: Embracing the resiliency of deep neural networks. In *MICRO*, pages 256–269, 2020.

[9] M. Sun, Z. Liu, A. Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint*, arXiv:2306.11695, 2023.

[10] Y. Xue, Y. Liu, L. Nai, and J. Huang. V10: Hardware-assisted npu multi-tenancy for improved resource utilization and fairness. In *ISCA*, pages 1–15, 2023.

[11] L. Yin, A. Ghazizadeh, S. Tian, A. Louri, and H. Zheng. Polyform: A versatile architecture for multi-dnn execution via spatial and temporal acceleration. In *ICCD*, pages 166–169. IEEE, 2023.