2025 62nd ACM/IEEE Design Automation Conference (DAC) | 979-8-3315-0304-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/DAC63849.2025.11132628

# Pipirima: Predicting Patterns in Sparsity to Accelerate Matrix Algebra

Ubaid Bakhtiar, Donghyeon Joo, and Bahar Asgari
*University of Maryland, College Park*
{ubaidb, dhjoo98, bahar}@umd.edu

***Abstract*—While sparsity, a feature of data in many applications, provides optimization opportunities such as reducing unnecessary computations, data transfers, and storage, it causes several challenges, too. For instance, even in state-of-the-art sparse accelerators, sparsity can result in load imbalance; a performance bottleneck. To solve such challenges, our key insight is that if while reading/streaming compressed sparse matrices we can quickly anticipate the locations of the non-zero values in a sparse matrix, we can leverage this knowledge to accelerate processing sparse matrices. To enable this, we propose Pipirima, a lightweight prediction-based sparse accelerator. Inspired by traditional branch predictors, Pipirima uses resource-friendly simple counters to predict the patterns of non-zero values in the sparse matrices. We evaluate Pipirima based on sparse matrix vector multiplication (SpMV) and sparse matrix-dense matrix multiplication (SpMM) kernels on CSR compressed matrices derived from both scientific computing and transformer models. On average, our experiments show $6\times$ and $4\times$ speed up over Tensaurus for SpMM and SpMV, respectively on SuiteSparse workload. Pipirima also shows $40\times$ speed up over ExTensor for SpMM. We achieve $8.3\times$, $48.2\times$ over Tensaurus and ExTensor in lesser sparse transformer workloads. Piprima consumes $5.621mm^2$ area and $544.93mW$ power using 45nm technology with predictor related components as the least expensive ones.**

## I. Introduction

Over the years, sparse matrices have become crucial in various fields such as scientific computing, bioinformatics, genomics, computer vision, transformers and recommendation systems. While sparsity offers opportunities to eliminate unnecessary computations, storage, and data movement, the effectiveness of handling sparsity depends on several factors including compression methods [1] such as the frequently-used compressed sparse row (CSR) or more specific formats such as diagonal (DIA) [26], block CSR/CSC [33], list of lists (LIL) [27], and Ellpack (ELL) [15] that suit specific sparsity patterns or certain implementations. For brevity, this work targets CSR format as it stands out as the widely adopted format. However, CSR also comes with some limitations as we explain throughout this paper.

To efficiently run sparse problems, several domain-specific architectures (DSAs) [3], [9], [18], [24], [25], [29], [32], [37] have been developed, targeting sparsity in various domains such as scientific computing [2], [4], [5], [8], [10], [14], [17], [23], [28] DNNs [7], [13], [19]–[21], [34], [36] and sparse attention mechanisms [22], [35], some focusing on very specific aspects such as dataflow architectures, or implementing sparse DNNs on dense systolic arrays [11], [16]. In addition to the *domain-specific* studies, *kernel-specific* accelerators have been proposed to target common sparse matrix algebra, such as sparse matrix-vector multiplication (SpMV), sparse matrix-matrix multiplication (SpMM) sparse-sparse matrix multiplication (SpGEMM) that offer methods ranging from new microarchitectural supports for sparsity to enhancing memory bandwidth utilization. ExTensor [12], for instance, proposes a hierarchical elimination of computation in sparse environments by identifying multiplication cases where both operands are non-zero, thus avoiding unnecessary data transfers. Some of the prior studies integrate compression format into their co-optimization cycle. For instance, Tensaurus [30] employs a novel format for accessing sparse data in a vectorized and streaming manner, optimizing memory bandwidth usage by splitting dense vectors into tiles. The exploration of recent sparse accelerators underscores two major challenges. First, despite structured sparsity in matrices, these accelerators often perform standard operations without leveraging these patterns for quicker computations. Second, fine-grained load imbalance, as a result of unstructured sparsity, hinders the full utilization of parallelism capabilities as also discussed in [31].

To overcome these challenges with minimal hardware and latency overhead and without relying on extensive preprocessing, we introduce Pipirima[1], a novel approach using a lightweight, rapid sparsity pattern predictor based on counter-based prediction (CBP), similar to branch predictors. Pipirima uses a 1-bit counter to identify matrix structural patterns, enabling quick identification and omission of standard operations. In non-diagonal matrices, this prediction method forecasts the number of non-zero (NNZ) values per row in the streaming blocks to improve load balance and enhance fine-grained parallelism. Unlike traditional CSR-based computation, Pipirima distributes values based on the state of the predictor, and includes a straightforward inspector for the verification process for prediction accuracy and tracking the states. We also propose a novel architecture with Sigma-like [24] adder tree and arrangement of multipliers that uses the predictions to do the SpMM and SpMV operations. We evaluate Pipirima using a cycle-accurate simulator, conducting experiments on the CSR-based random and diagonal matrices from SuiteSparse dataset [6], transformer models, and synthetic random matrices. Our experiments demonstrated a significant speed up compared to baseline sparse accelerators; ExTensor [12] and Tensaurus [30] at the expense of very small memory and computational overhead of CBP predictors and Prediction Inspectors. The area and power consumption of

[1]Pipirima is a star in the zodiac constellation of Scorpius.

Pipirma are also very small, with Prediction Inspectors as the least expensive components.

## II. Motivation & Key Insight

***Challenges & Acceleration Opportunities:*** We explore the challenges and opportunities of recent SpMM accelerators from two angles. ***(1)*** *The first issue is that even if the sparse matrices manifest structured sparsity, such as diagonal, the SpMM accelerators still undergo standard operations instead of taking shortcuts to quicker results.* ***(2)*** *The second issue concerns the fine-grained load imbalance, referring to the uneven distribution of NNZ values among units of on-chip memory and computational units.* This imbalance prevents sparse matrix algebra from maximizing fine-grained parallel capabilities of hardware. Our key insight to solve the aforementioned challenges is to use simple prediction mechanisms to first predict a categorical pattern of sparsity in a matrix and accordingly skip some computations and second, if the matrix does not have a pattern, use prediction to enable ideal load balancing. In the following we provide more details about our key insights to solve each of the challenges.

*1)* ***D/R Matrix Predictor:*** Given the structural peculiarity of diagonal matrices and since the indices of NNZ values in a diagonal matrix are known, we can bypass some steps of processing a sparse matrix (e.g., reading offsets and column indices when CSR format is used) and directly feed the input values into the compute unit.

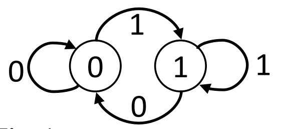


**Fig. 1: State diagram for the D/R Matrix Predictor–** States 0 and 1 indicate diagonal and random.

To quickly identify the structure of a matrix blocks, we use a single bit for prediction. As shown in Figure 1, D/R Matrix Predictor includes two states: *(i) Random Matrix (ii) Diagonal Matrix*, where the prediction problem is mapped to a binary problem by representing ***0*** for random matrix and ***1*** for diagonal matrix. The predictor works by observing the structure of the blocks – the current block is predicted to be the same as the previous one.

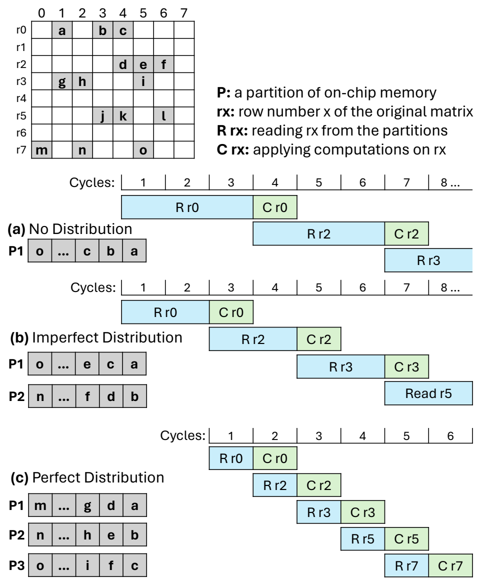


**Fig. 2: Distribution matters– (a)** Serial placement of NNZ values, resulting in bottleneck. **(b)** Sub-par parallelism due to NNZ distribution among two memory partitions. **(c)** Maximum parallelism due to perfect NNZ distribution.

*2)* ***NNZ/row Predictor:*** Before going through our solution to the second challenge, we provide more details about the challenge of fine-grained load imbalance starting with reviewing the importance in load distribution.

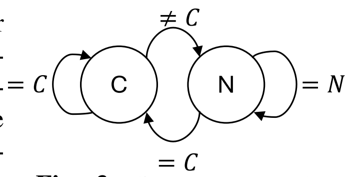
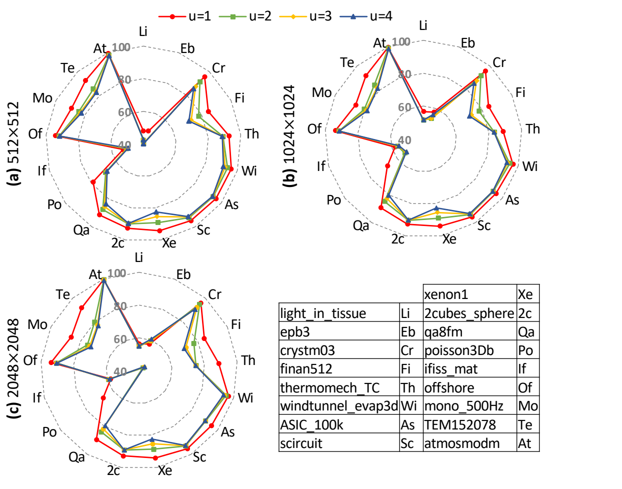


**Fig. 3: State diagram for the NNZ/row Predictor–** States C and N indicate the current and new number of non-zero/row.

***Why distribution is important?*** Here, we assume that we have a streaming accelerator including a pipeline buffer. To process a sparse matrix such as the one shown in Figure 2, for each row, we need to first *read* the NNZ elements from the on-chip buffer (i.e., `R rx` in Figure 2) and then perform the *computation* of that row (i.e., `C rx` in Figure 2), where `R rx` and `C rx` are pipelined. Given such a pipeline, Figure 2 compares three cases: **(a)** having only one partition of on-chip memory, which serializes reading the NNZ values, **(b)** having more than one partition of on-chip memory but an imperfect distribution of NNZ values, which reduces the `R rx` time but does not completely solve the problem, and **(c)** a perfect distribution of NNZ values to as many partitions as they need which minimizes the `R rx` and results in a more balance pipeline. While this figure illustrates the importance of load balance given a dot-product based computation, a row-wise or outer-product based implementation suffer from a similar load-imbalance issue because of sparsity.

***Our key insight to achieve the perfect distribution with minimal costs.*** Pipirma enables the optimized implementation shown in Figure 2c. Our key insight to enable this is to use a simple state machine shown in Figure 3 that predicts the NNZ values for the current block based on history of the previous blocks (like branch predictors). Using such a simple predictor, by just reading a single counter, as we stream values



| | | | |
|---|---|---|---|
| | | xenon1 | Xe |
| light_in_tissue | Li | 2cubes_sphere | 2c |
| epb3 | Eb | qa8fm | Qa |
| crystm03 | Cr | poisson3Db | Po |
| finan512 | Fi | ifiss_mat | If |
| thermomech_TC | Th | offshore | Of |
| windtunnel_evap3d | Wi | mono_500Hz | Mo |
| ASIC_100k | As | TEM152078 | Te |
| scircuit | Sc | atmosmodm | At |

**Fig. 4: Row homogeneity % for the last “u” neighbours of SuiteSparse workloads**

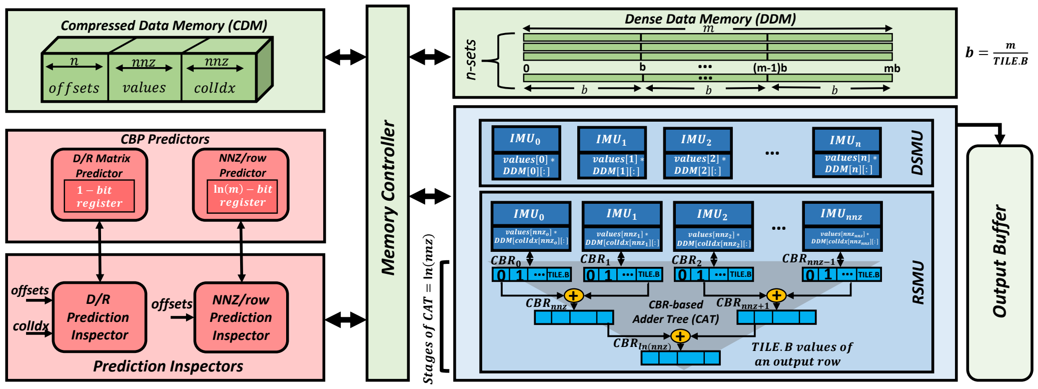

Compressed Data Memory (CDM)
$n$
$nnz$
$nnz$
$offsets$
$values$
$colIdx$
CBP Predictors
D/R Matrix Predictor
$1-bit$ $register$
NNZ/row Predictor
$\ln(m)-bit$ $register$
offsets
colIdx
D/R Prediction Inspector
offsets
NNZ/row Prediction Inspector
Prediction Inspectors
Memory Controller
Stages of CAT = $\ln(nnz)$
Dense Data Memory (DDM)
$m$
n-sets
0
b
(m-1)b
mb
$b$
$b$
$b$
$b = \frac{m}{TILE.B}$
$IMU_0$ $values[0]*DDM[0][:]$
$IMU_1$ $values[1]*DDM[1][:]$
$IMU_2$ $values[2]*DDM[2][:]$
$IMU_n$ $values[n]*DDM[n][:]$
DSMU
$IMU_0$ $values[nnz_0]*DDM[colIdx[nnz_0]][:]$
$IMU_1$ $values[nnz_1]*DDM[colIdx[nnz_1]][:]$
$IMU_2$ $values[nnz_2]*DDM[colIdx[nnz_2]][:]$
$IMU_{nnz}$ $values[nnz_{nnz}]*DDM[colIdx[nnz_{nnz}]][:]$
$CBR_0$
$CBR_1$
$CBR_2$
$CBR_{nnz-1}$
0 1 ••• TILE.B
$CBR_{nnz}$
CBR-based Adder Tree (CAT)
$CBR_{nnz+1}$
$CBR_{\ln(nnz)}$
$TILE.B$ values of an output row
RSMU
Output Buffer

**Fig. 5: Architecture of Pipirima–** The memory controller sends the input data based on the predictions from CBR and forwards it to the Prediction Inspectors and Compute Units.

and indices, we distribute them evenly.

***Why does the proposed solution work?*** In Figure 4, we run experiments on some matrices under different partition sizes from the SuiteSparse dataset to see how many rows have the same NNZ values as their last $u$ neighbors. We see that on average 78-85% of the rows are homogeneous to their last $u$ neighbors in terms of NNZ values. Notably, when $u = 1$, meaning only the last neighbor is considered, the resemblance is most pronounced. This finding motivates us to exploit the structural coherence of sparse matrices to ensure perfect distribution and acceleration of computations associated with sparse matrix algebra.

## III. Pipirima

### A. *High-level Overview of Pipirima*

In this work, we focus on accelerating sparse algebra computations assuming that all the essential data is available within the on-chip memory. Therefore, we do not delve deeply into data movement between off-chip memory and the on-chip network. We implement a tile-based mechanism in our compute units, which is pivotal for facilitating fine-grained parallelism in each multiplication operation, making Pipirima flexible to adapt to different configurations. (details in Section IV-B). Figure 5 shows the architecture of Pipirima, the details of which are broken into three groups: memory components, prediction components, and computational components (shown in green, red, blue in Figure 5, respectively).

### B. *Memory Components*

*1)* ***Compressed Data Memory (CDM):*** Compressed Data Memory (CDM) is an on-chip memory that is responsible for holding the sparse input data compressed in CSR format. CDM has a predominantly read-oriented role in the context of Pipirima. Upon receiving a command from the memory controller, the CDM retrieves and dispatches the requested compressed data to the designated unit within the architecture.

*2)* ***Dense Data Memory (DDM):*** Dense Data Memory (DDM) is responsible for storing the dense matrix or vector operand. For an $n \times m$ matrix partition, DDM has $n$ sets of $m$-size blocks. These blocks are split into *TILE.B* number of tiles. Dense matrix is stored in row-major format in these sets. This improves the spatial locality of a dense matrix such that when referring to a row-vector and its tile, the compute units can always go to the specific set without any memory miss.

### C. *Prediction Components*

*1)* ***CBP Predictor:*** The D/R Matrix Predictor and NNZ/row Predictor reside in this unit. Each of these predictors have one register that is responsible for holding the current state. The D/R Matrix Predictor has a 1-bit register as it represents only two states. For an $n \times m$ matrix, the NNZ/row Predictor consists of a $ln(m)$-bit register, which is an insignificant hardware cost associated with our CBP predictors. The registers in CBP predictors receive an input from their corresponding prediction inspectors to update the states.

*2)* ***Prediction Inspector:*** To keep the states of the predictor updated, we need to verify whether a prediction was correct or not and change the state accordingly if needed. In this section, while our current emphasis lies in detailing the prediction verification steps tailored for CSR format within Pipirima, the high-level approach holds significant potential for seamless adaptation to other compressed formats. Following two prediction inspectors comprise Pipirima:

*D/R Prediction Inspector–* In a diagonal matrix the consecutive values in the column indices and row offset arrays have a difference of 1 and the size of these arrays is equal to $n$ for an $n \times n$ matrix. Pipirima exploits these simple properties to inspect the predictions. The prediction verification process is performed in a pipeline with three stages; compute, compare, and output In the compute stage, the subtraction operation is done and the compare stage compares it with the D/R Matrix Prediction. If the comparison fails, this signifies a misprediction and Pipirima does the following:

- If prediction=1 (i.e., diagonal), it flushes the pipeline, resets the values sent to compute units, sets the state register to 0 (random), triggers the NNZ/row Predictor and gets the correct data from the memory for the Prediction Inspector.
- If prediction=0 (i.e., random), it flushes the pipeline, resets the values sent to the NNZ/row Predictor, sets the state register to 1 (diagonal), signals the memory controller, and sends values to the compute unit directly.

In the case of correct prediction, Pipirima continues with its normal working without any changes in the state register.

*NNZ/row Prediction Inspector–* The mechanism of this inspector is simple, too, as the number of non-zeros for a row in a block can be calculated easily based on the offsets. If the comparison is true, this signifies a correct prediction, hence no change in the state. A misprediction by the the NNZ/row predictor does not cause any changes in the functionality as it only defines the load distribution. Therefore, an NNZ/row misprediction only leads to a less than optimal performance and detecting such a misprediction only requires updating the state for the next predictions.

## *D. Computational Components*

*1) **If-Mul Units (IMU):*** This unit is responsible for multiplication but even though the dense matrix is not sparse yet there can be few zero values that result in wasted number of multiplication cycles. To avoid this inefficiency, we introduce a condition using a comparator and multiplexer, to check if the values from DDM are non-zero or not. If it is equal to zero, it is pushed to the CBR, else the multiplication units (MU) would be triggered. IMU does multiplications in parallel fashion depending on the tiling factor (*TILE.B*) of dense matrix B. IMU executes *TILE.B* number of multiplications in each step and forwards the data to the CBR, whereas the total number of steps are $b = \frac{m}{TILE.B}$. Pipirima is a versatile hardware that can be configured to implement SpMV as well as SpMM using dot product or row-wise product. For SpMV, each IMU will do one $values[nnz_i^j] \times DMM[0][colIdx[nnz_i^j]]$ operation. In the case of SpMM, each IMU does one $values[nnz_i^j] \times DMM[colIdx[nnz_i^j]][k]$ operation.

*2) **Contiguous Block Registers (CBR):*** The partial outputs are stored in CBR, the size of which is insignificant as it only holds $TILE.B$ number of partial outputs. As shown in Figure 5, each IMU has a corresponding CBR where data is placed contiguously to improve the spatial locality of CBR for addition operations. This is for random sparse matrices only as the output of an SpMM with a diagonal matrix does not result in‘ partial outputs.

*3) **Diagonally Sparse Multiplier Unit (DSMU):*** In case of diagonal prediction from D/R Matrix Predictor, the memory controller sends the values array data to the DSMU. In the case of diagonal sparsity, the multiplication boils down to $values[nnz_i] \times DDM[i][:]$ multiplication for each IMU. There is no need for adders in DSMU. Hence, the output of multiplications is sent directly to the output buffers.

*4) **Randomly Sparse Multiplier Unit (RSMU):*** For random sparse matrices, there can be multiple non-zero values per row. The number of non-zero values are already known to the compute unit by the virtue of NNZ/row Predictor. Based on the prediction, RSMU reads that number of non-zero values equal to the prediction and allocate each of them to an IMU. An IMU multiplies the $i^{th}$ non-zero of $j^{th}$ row with the corresponding vector from DDM (i.e., $values[nnz_i^j] \times DDM[colIdx[nnz_i^j]][:]$), and stores the result in the corresponding block in CBR. For both DSMU and RSMU, we harness fine-grain parallelism by allowing each IMU to do *TILE.B* number of multiplication operations at a time. We can observe that Pipirima informed by the NNZ values per row (due to the prediction) can now employ multiple IMUs concurrently and does not have to segregate offsets load and values/indices load.

**TABLE I:** Pipirima configurations for SpMMs and SpMV.

| | | CFG-1 | CFG-2 | CFG-3 |
|---|---|---|---|---|
| | | SpMM | | SpMV |
| Hardware Features | | Row-wise Product | Dot-product | |
| TILE.B | | 4 | 4 | 1 |
| RSMU | Count# | 1 | 4 | 1 |
| IMU | Count# | 10 | 40 | 10 |
| | Mul-ops/step | TILE.B=4 | 1 | 1 |
| D/R Matrix Predictor | Register | 1-bit | | |
| NNZ/row Pr. | Register | $ln(Partition_{size})$ E.g.: | $512B \times 512B$: 9-bit<br>$1KB \times 1KB$: 10-bit<br>$2KB \times 2KB$: 11-bit | |
| CBR | Count# | 19 | 76 | 19 |
| | Size/CBR | 16B | 4B | 4B |
| | Total Size | 320B | 320B | 80B |
| CDM | Size | 64KB | | |
| DDM | Size | 128KB | | |
| Area ($mm^2$) | | 5.621 | 5.621 | 4.925 |
| Power Consumption ($mW$) | | 544.93 | 544.93 | 445.42 |

*5) **CBR-based Adder Tree (CAT):*** IMUs are doing $TILE.B$ number of multiplication operations in each step. These partial outputs are pushed to the corresponding CBR. We utilize a SIGMA-like [24] CBR-based Adder Tree (CAT), designed to accumulate results from adjacent CBR units. CAT runs in parallel to the IMUs, concealing the partial outputs accumulation latency. Figure 5 shows CAT inside an RSMU. CAT has $ln(nnz)$ number of stages – if a row has four non-zeros, CAT will have two adder stages. The contiguous placement of data within CBRs facilitates addition across different CBRs without the need for index matching, hence streamlining the process. The final outputs of the adder tree are $TILE.B$ elements of the $j^{th}$ row of the output matrix and are moved to the output buffer.

## IV. Experimental Setup

### *A. Workloads*

**SuiteSparse Dataset.** We select a combination of random and diagonal matrices from SuiteSparse [6], few listed in Figure 4 with varying patterns of sparsity. The size of matrices range from $6M \times 6M$ to $1k \times 1k$. We break down, stream, and process these matrices into $512 \times 512$, $1024 \times 1024$, and $2048 \times 2048$ partitions.

**Synthetic Matrices.** To analyze the behavior of Pipirima on highly dense matrices to more precisely investigate its impact on metrics such as misprediction, we also use synthetic random matrices of size $4096 \times 4096$ with varying density.

**Transformer Workloads.** To evaluate Pipirima for a wider range of applications, we use Sanger's [22] code to create sparse matrices from the multi-head operations of transformer BERT trained on SQuAD v1.1. These matrices have a higher density (i.e., up to 32%) and are smaller than the SuiteSparse matrices. Therefore, here we use smaller partition sizes of $16 \times 16$, $32 \times 32$ and $64 \times 64$.

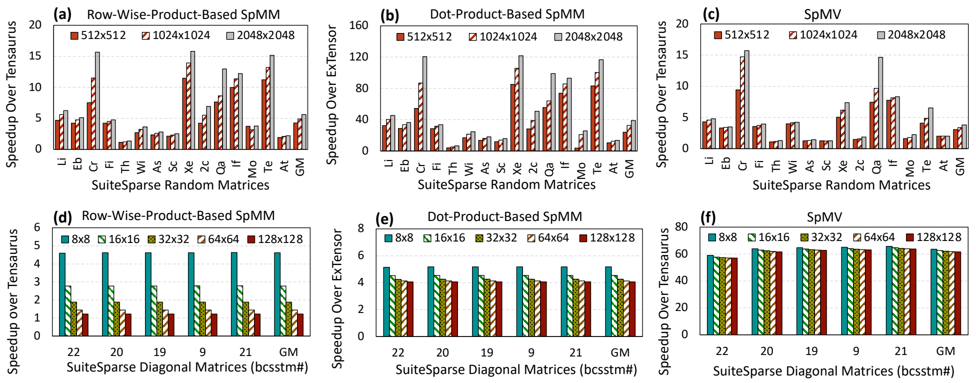


Fig. 6: **Speed up for SuiteSparse matrices–** (**a**), (**b**), and (**c**): Random Matrices; (**d**), (**e**) and (**f**): Diagonal Matrices

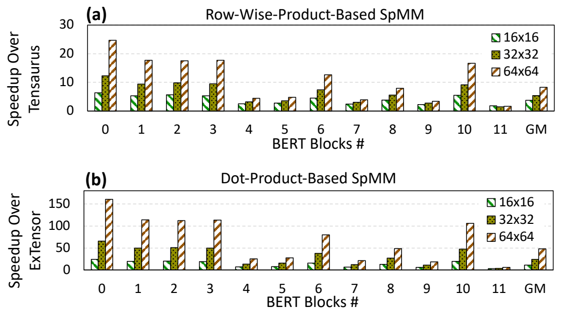


**Fig. 7: Speed up for Transformer matrices–** (**a**) row-wise product and (**b**) dot-product.

*B. Baselines & Simulation Setup*

We use ExTensor [12] as our baseline for dot-product-based SpMM; and Tensaurus [30] for row-wise-product-base SpMM as well as SpMV. In both of these baselines, we assume that the required data for a given computation resides in the on-chip memory. They are implemented with the same configurations as presented in their respective papers. We use a cycle-accurate simulator to model both Pipirima and baselines. We implement RSMU, DSMU, Prediction Inspector, and CBP predictor in RTL using Verilog and synthesize them using Synopsys Design Compiler in TSMC 45nm library. We use CACTI 7.0 to estimate the area, power, and latency of the memory components. We are experimenting Pipirima on SpMM using row-wise and dot product as well as SpMV kernel. Table I outlines the different configurations of Pipirima. It also shows the area and power consumption.

## V. Evaluations and Results

***Speed up:*** To examine effective acceleration, improving which is the driving motivation for Pipirima, Figure 6 represents the speed up obtained for the SuiteSparse matrices and Figure 7 depicts the speed up for the multi-attention layers of BERT. These matrices are denser than SuiteSparse matrices, giving us an opportunity to evaluate Pipirima on less sparse matrices. One of the main reasons for achieving ***performance improvements over baselines*** is the parallel computational operations corresponding to each non-zero value. *The prior knowledge of matrix partition structure – diagonal or random and NNZ values per row, allows us to distribute the multiplication operations among different IMUs– key benefit of predictions.* Pipirima also uses tiling that helps to further parallelize multiplication operations within each IMU at the expense of extra multipliers. Tensuarus also uses tiling mechanism to accelerate computations. This is the reason the speed up improvement over Tensaurus is not as much as ExTensor.

***Effects of Mispredictions:*** Mispredictions are inherent in predictive systems and can impact system performance. Mispredictions affect the number of memory accesses and computations performed by Piprima. Figures 10 a, b, and c show the relationship between these quantities on SuiteSprase matrices with varying partition sizes. We set up a ratio to study this behavior.

A ratio less than 1 signifies more overhead of prediction components, and greater than 1 signifies more overhead in traditional decompression.

To analyze the behavior on fixed size matrix, with varying density, we are using synthetic random matrices with varying partition sizes. Figures 10 d, e, and f represent a uniform behavior between mispredictions and these overheads. These graphs also depict the same behavior. Figure 8 investigates the relationship between the density and misprediction percentage at different partition sizes. To ensure fairness in the comparison, synthetic random matrices with fixed size and varying partition sizes have been used for this experiment. From these results, we extract two important trade-offs. First, with the increasing density, percentage mispredictions increase due to the increased probability of different NNZ values in consecutive rows and decreased row homogeneity score. Second, the percentage misprediction increases with the increasing partition sizes too. This is due to the increased number of allowed states for the NNZ/row Predictor state register, making the system vulnerable to more mispredictions.

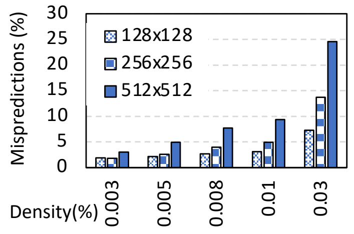


**Fig. 8: Mispredictions (lower is better) vs. Density for Synthetic matrices**

***Overhead Analysis:*** CBP predictors are cheap and comprise only one register. However, the Prediction Inspectors incur memory and computational overhead. Figure 9 shows the

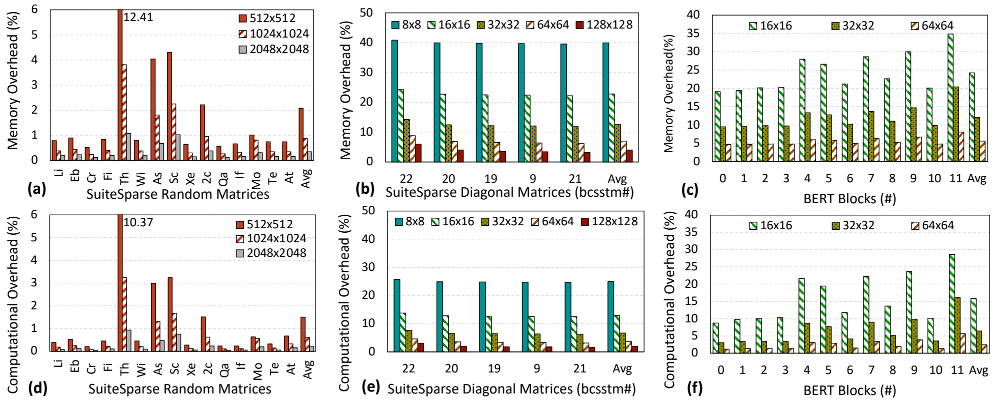


Fig. 9: **Overheads of prediction inspector– (a), (b),** and **(c)** Memory overhead, and **(d), (e),** and **(f)** Computational overhead

memory and computational overhead for different datasets. The overhead increases if the matrices are dense (BERT blocks) or if they are small in size. (diagonal matrices) We also observe that these overheads diminishes as the partition size increases in random matrices. The reason is fewer predictions are needed for bigger partition sizes and hence, fewer instances to be verified. For some cases (e.g. Th, Sc, As), the overhead is relatively bigger, because of their highly random distribution of non-zeros and smaller number of multiplication operations. Moreover, we observe that the Prediction Inspectors consume the least share of total area and power, that is, $0.13\%, 0.15\%$ of the total area and $0.44\%, 0.54\%$ of the total power for CFG-1, CFG-2 and CFG-3 respectively.

## VI. Related Work

In recent years, several DSAs has been developed to tackle the challenges of sparse computations in various domains. Since sparsity is also prominent in DNNs, several DSAs address specific challenges in this domain, such as sparse attention mechanisms [22] and efficient execution on dense systolic arrays [11], [16]. Other DSAs atarget specific kernels icnluding SpMV, SpMM, and SpGEMM. Sigma [24], for instance, addresses irregular and unstructured SpGEMM operations and employs an adder tree network for efficient partial sum accumulation. ExTensor reduces wasted computations by identifying multiplication cases with non-zero operands ahead of time [12]. SpArch [37] optimizes the data locality of both input and output matrices in the SpMM kernel, employing a highly parallelized streaming-based merger and a condensed data representation to reduce redundant access to zero values. MatRaptor [29], a SpGEMM accelerator, uses row-wise products for high data reuse and introduces a new compression format, $C^2SR$. This compression format provides information about the NNZ values per row. However, load-balancing in MatRaptor still relies on round-robin allocation which does not always guarantee a perfect load balance. Another row-wise product-based study [18] proposes a counter-based matrix tiling scheme specific to CSR-compressed format to enhance parallelism in SpMM.

## VII. Conclusion

This paper proposed Pipirima, a novel approach to accelerate sparse matrix algebra. By effectively predicting the structural characteristics of matrices, Pipirima enabled the tile based parallel SpMV and SpMM kernels. Extensive experimentation demonstrated Pipirima's remarkable acceleration capabilities compared to prior sparse accelerators. Importantly, Pipirima achieved these gains while maintaining minimal memory and computational overhead in its prediction components, making it a valuable asset for a wide range of applications.

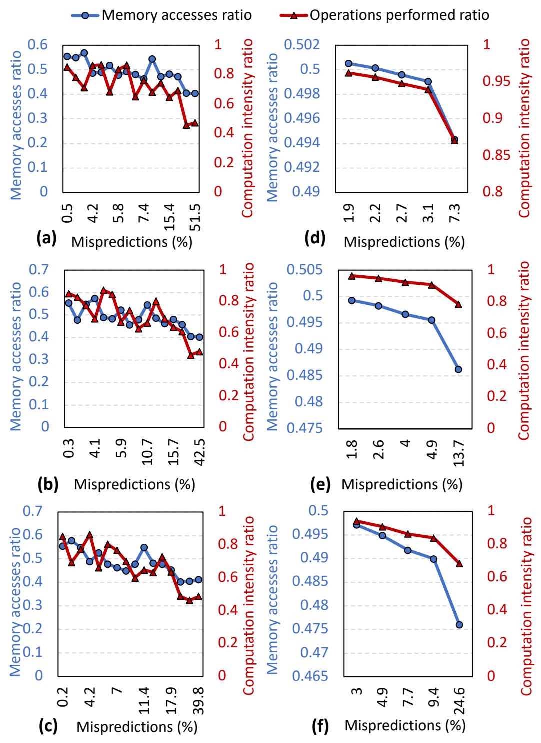


**Fig. 10: Memory accesses and computation intensity ratio (higher is better) vs. Mispredictions (%)– (a), (b),** and **(c)** represent the SuiteSparse non-diagonal matrices with partition sizes 512, 1024, 2048; **(d), (e),** and **(f)** represent the synthetic random matrices with partitions of 128, 256, 512 respectively.

## Acknowledgments

Ubaid Bakhtiar is supported by the U.S. DoE, Office of Science, Office ACSR, the ECRP program, under Award Number DE-SC0024079. Donghyeon Joo is supported by the NSF PPoSS program, under Award Number 2316177.

## References

[1] B. Asgari, R. Hadidi, J. Dierberger, C. Steinichen, A. Marfatia, and H. Kim, "Copernicus: Characterizing the performance implications of compression formats used in sparse workloads," in *IISWC*. IEEE, 2021, pp. 1–12.

[2] B. Asgari, R. Hadidi, T. Krishna, H. Kim, and S. Yalamanchili, "Alrescha: A lightweight reconfigurable sparse-computation accelerator," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 249–260.

[3] U. Bakhtiar, H. Hosseini, and B. Asgari, "Acamar: A dynamically reconfigurable scientific computing accelerator for robust convergence and minimal resource underutilization," in *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2024, pp. 1601–1616.

[4] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "An sram-based accelerator for solving partial differential equations," in *2019 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2019, pp. 1–4.

[5] T. Chen, J. Botimer, T. Chou, and Z. Zhang, "A 1.87-mm 2 56.9-gops accelerator for solving partial differential equations," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1709–1718, 2020.

[6] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, p. 1, 2011.

[7] C. Deng, Y. Sui, S. Liao, X. Qian, and B. Yuan, "Gospa: An energy-efficient high-performance globally optimized sparse convolutional neural network accelerator," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1110–1123.

[8] B. Feinberg, U. K. R. Vengalam, N. Whitehair, S. Wang, and E. Ipek, "Enabling scientific computing on memristive accelerators," in *The International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 367–382.

[9] A. Gerami and B. Asgari, "Gust: Graph edge-coloring utilization for accelerating sparse matrix vector multiplication," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2024.

[10] N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis, "Energy-efficient hybrid analog/digital approximate computation in continuous time," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 7, pp. 1514–1524, 2016.

[11] 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 *Proceedings of the 34th ACM International Conference on Supercomputing*, 2020, pp. 1–12.

[12] K. Hedge, H. Asghari, M. Pellauer, N. Crago, A. Jaleel, E. Solomonik, J. Emer, and C. W. Flectcher, "Extensor: An accelerator for sparse tensor algebra," in *2019 International Symposium on Microarchitecture (MICRO-52)*. IEEE/ACM, 2019, pp. 319–333.

[13] C.-T. Huang, "Ringcnn: Exploiting algebraically-sparse ring tensors for energy-efficient cnn-based computational imaging," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1096–1109.

[14] Y. Huang, N. Guo, M. Seok, Y. Tsividis, K. Mandli, and S. Sethumadhavan, "Hybrid analog-digital solution of nonlinear partial differential equations," in *MICRO*. IEEE, 2017, pp. 665–678.

[15] D. R. Kincaid, T. C. Oppe, and D. M. Young, "Itpackv 2d user's guide," Texas Univ., Austin, TX (USA). Center for Numerical Analysis, Tech. Rep., 1989.

[16] H. Kung, B. McDanel, and S. Q. Zhang, "Packing sparse convolutional neural networks for efficient systolic array implementations: Column combining under joint optimization," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 821–834.

[17] J. Kung, Y. Long, D. Kim, and S. Mukhopadhyay, "A programmable hardware accelerator for simulating dynamical systems," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 403–415, 2017.

[18] J. H. Lee, B. Park, J. Kong, and A. Munir, "Row-wise product-based sparse matrix multiplication hardware accelerator with optimal load balancing," *IEEE Access*, vol. 10, pp. 64 547–64 559, 2022.

[19] G. Li, W. Xu, Z. Song, N. Jing, J. Cheng, and X. Liang, "Ristretto: An atomized processing architecture for sparsity-condensed stream flow in cnn," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1434–1450.

[20] S. Li, E. Hanson, X. Qian, H. H. Li, and Y. Chen, "Escalate: Boosting the efficiency of sparse cnn accelerator with kernel decomposition," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 992–1004.

[21] H. Lu, L. Chang, C. Li, Z. Zhu, S. Lu, Y. Liu, and M. Zhang, "Distilling bit-level sparsity parallelism for general purpose deep learning acceleration," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 963–976.

[22] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[23] J. Mu and B. Kim, "29.2 a 21× 21 dynamic-precision bit-serial computing graph accelerator for solving partial differential equations using finite difference method," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 406–408.

[24] E. Qin, A. Samajdar, H. Kwon, V. Nadella, S. Srinivasan, D. Das, B. Kaul, and T. Krishana, "Sigma: A sparse and irregular gemm accelerator with flexible interconnects for dnn training," in *2020 IEEE Internation Sumposium on High Performance Computer Architecture*. IEEE, 2020, pp. 58–70.

[25] D. Ramchandani, B. Asgari, and H. Kim, "Spica: Exploring fpga optimizations to enable an efficient spmv implementation for computations at edge," in *2023 IEEE International Conference on Edge Computing and Communications (EDGE)*. IEEE, 2023, pp. 36–42.

[26] Y. Saad, *Iterative methods for sparse linear systems*. siam, 2003, vol. 82.

[27] SciPy, "List-of-list sparse matrix," 2020, [Online; accessed April-2020].

[28] G. Singh, D. Diamantopoulos, C. Hagleitner, J. Gómez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, "Nero: A near high-bandwidth memory stencil accelerator for weather prediction modeling," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 9–17.

[29] N. Srivastava, H. Jin, J. Liu, D. Albonesi, and Z. Zhang, "Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 766–780.

[30] N. Srivastava, H. Jin, S. Smith, H. Rong, D. Albonesi, and Z. Zhang, "Tensaurus: A versatile accelerator for mixed sparse-dense tensor computations," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 689–702.

[31] J. Tong, A. Itagi, P. Chatarasi, and T. Krishna, "Feather: A reconfigurable accelerator with data reordering support for low-cost on-chip dataflow switching," 2024. [Online]. Available: https://arxiv.org/abs/2405.13170

[32] C. K. Vadlamudi and B. Asgari, "Electra: Eliminating the ineffectual computations on bitmap compressed matrices," *IEEE Computer Architecture Letters*, 2024.

[33] R. W. Vuduc and H.-J. Moon, "Fast sparse matrix-vector multiplication by exploiting variable block structure," in *International Conference on High Performance Computing and Communications*. Springer, 2005, pp. 807–816.

[34] Y. Wang, C. Zhang, Z. Xie, C. Guo, Y. Liu, and J. Leng, "Dual-side sparse tensor core," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1083–1095.

[35] A. Yazdanbakhsh, A. Moradifirouzabadi, Z. Li, and M. Kang, "Sparse attention acceleration with synergistic in-memory pruning and on-chip recomputation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 744–762.

[36] S. Zeng, Y. Lin, S. Liang, J. Kang, D. Xie, Y. Shan, S. Han, Y. Wang, and H. Yang, "A fine-grained sparse accelerator for multi-precision dnn," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2019, pp. 185–185.

[37] Z. Zhang, H. Wang, S. Han, and W. J. Dally, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 261–274.