FullSparse: A Sparse-Aware GEMM Accelerator with Online Sparsity Prediction

Jiangnan Yu
Fudan University
Shanghai, China
jiangnanyu21@m.fudan.edu.cn

Yang Fan
Fudan University
Shanghai, China
yangfan@fudan.edu.cn

Hanfei Wang
Fudan University
Shanghai, China
wanghahui57@gmail.com

Yuxuan Qiao
Fudan University
Shanghai, China
23212020135@m.fudan.edu.cn

Zheng Wu
Fudan University
Shanghai, China
wuz20@fudan.edu.cn

Xiankui Xiong
State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation
Shanghai, China
xiong.xiankui@zte.com.cn

Xiao Yao
State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation
Shanghai, China
yao.xiao1@zte.com.cn

Haidong Yao
State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation
Shanghai, China
yao.haidong@zte.com.cn

Yecheng Zhang
State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation
Shanghai, China
zhang.yecheng@zte.com.cn

ABSTRACT

Leveraging sparsity optimizes storage and computation for resource-constrained devices in Deep Learning Neural Networks (DNNs). While neural networks naturally incorporate sparsity through operations like ReLU and quantization, diverse sparsity levels (0.2% to 99%) pose challenges for the design of computational units. In this paper, we provide an energy-efficient GEMM accelerator named FullSparse which is designed for diverse applications, accommodating varying sparsity levels in matrix multiplication (0.2% to 99%). This paper introduces three features for nuanced sparsity support: multi-sparsity control, predictive result sparsity, and a multi-sparsity-compatible PE array. Experimental evaluations affirm that our implementation while ensuring adaptability to sparsity, exhibits superior computational power comparable to the existing designs.

CCS CONCEPTS

• **Hardware → Application specific processors**.

KEYWORDS

Deep learning Accelerator, Sparsity, Neural network

ACM Reference Format:
Jiangnan Yu, Yang Fan, Hanfei Wang, Yuxuan Qiao, Zheng Wu, Xiankui Xiong, Xiao Yao, Haidong Yao, and Yecheng Zhang. 2024. FullSparse: A Sparse-Aware GEMM Accelerator with Online Sparsity Prediction. In *21st ACM International Conference on Computing Frontiers (CF '24), May 7–9, 2024, Ischia, Italy.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3649153.3649180

CF '24, May 7–9, 2024, Ischia, Italy

ACM ISBN 979-8-4007-0597-7/24/05
https://doi.org/10.1145/3649153.3649180

1 INTRODUCTION

Deep Neural Networks (DNNs) are foundational in machine learning, excelling in tasks like image classification, object recognition, and tracking. As mobile devices advance, there's a growing trend for resource-constrained devices to locally execute CNN applications, providing faster response times, reduced bandwidth, and enhanced data privacy [4][7]. However, the computational and memory demands of CNNs pose challenges for direct deployment on constrained systems.

Effectively leveraging sparsity can reduce storage and computation needs, making CNNs more suitable for resource-constrained devices. Neural networks (NNs) inherently exhibit natural sparsity, with ReLU operations zeroing out data in activations and many values in weights becoming zero after quantization. Pruning techniques further increase weight sparsity by eliminating unimportant data during network training. However, real-world NNs have a wide distribution of sparsity, ranging from 0.2% to 99%, posing challenges for sparsity handling. Traditional GPU/CPU solutions with sparse libraries like cuSPARSE struggle to handle irregular sparsity in CNNs.

Optimizing hardware accelerators for sparsity is crucial. Recent investigations in Application-Specific Integrated Circuit (ASIC) accelerators [1][9][11]have focused on sparsity intricacies in neural networks. Two categories have emerged: one exploits sparsity in computational frameworks, effectively reducing computational power consumption but accumulating storage overhead [3][8]. The other integrates sparsity into both computation and storage paradigms [12][5][2][10], storing networks in a sparse matrix

format to enhance energy efficiency. However, this method faces challenges when the network lacks substantial sparsity, leading to pronounced overhead in index storage. Designing specialized hardware accelerators is key to navigating these challenges, requiring further exploration for an optimal balance between computational efficiency and storage considerations.

While studies have explored sparsity in Neural Networks (NNs), understanding its nuances remains challenging. Previous research often oversimplifies networks into sparse or dense categories, overlooking the diverse sparsity distribution in real-world NNs (ranging from 0.02% to 99%). Handling this diversity is a significant challenge. Additionally, studies tend to exclusively focus on storing weights or activations in sparse formats, introducing irregularities in convolutional operations that impact output data. This irregularity poses challenges in memory bandwidth and silicon area overhead in hardware implementation, particularly with standard Static Random-Access Memory (SRAM) macros.

This paper presents FullSparse, a Convolutional Neural Network (CNN) accelerator addressing challenges in sparsity. FullSparse focuses on three key aspects: 1) sparsity coverage; 2) storage efficiency; and 3) computational efficiency. To achieve these goals, three innovations are introduced:

1) An online input data sparsity detector is proposed to detect activation sparsity and dynamically switch the accelerator state among three sparse modes. 2) A multi-sparsity-compatible set-associative Processing Element (PE) array efficiently processes convolutions with different sparsity, handling irregular output data using collision-free methods. 3) A framework with an online result sparsity predictor predicts results' sparsity and converts data storage formats among three sparse formats to enhance off-chip memory access efficiency.

The paper is organized as follows: Section II discusses CNN concepts and sparsity. Section III introduces motivation and architectural design. Section IV presents the online sparsity adaptor and result sparsity predictor. Section V details measurement results, emphasizing the wide-ranging sparsity-compatible capacity. Section VI concludes the paper.

2 SPARSITY IN NEURAL NETWORKS

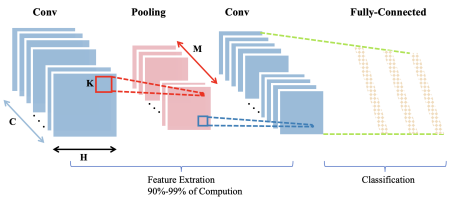

Figure 1: Typical DNN architecture overview.

Convolutional Neural Networks (CNNs), as depicted in Fig 1, are widely used for processing 2-dimensional data like images or videos. Comprising convolution, pooling, and fully connected layers, these networks focus on feature extraction and classification.

In Fig 2, sparsity in both activations and weights is evident. ReLU operations induce activation sparsity by nullifying negative activations, while weight sparsity results from pruning techniques

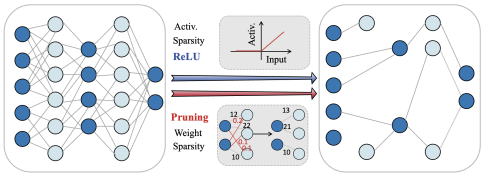

Figure 2: Sparsity in NNs.

[6],[13]. Training involves backpropagation, followed by the removal of weights below a threshold and retraining [6]. Recent work [13] formulates weight pruning as a non-convex optimization problem with combinatorial constraints specifying sparsity requirements.

2.1 Sparse Matrix Multiplication Algorithms

Sparse matrix multiplication (GEMM) relies on two key algorithms: the inner-product and outer-product algorithms, crucial for efficient computations.

The **inner-product algorithm** involves element-wise multiplication and accumulation of products from non-zero elements of two matrices. While effective in some cases, its efficiency may decrease with highly sparse matrices, resulting in wasted computational resources.

In contrast, the **outer-product algorithm** offers an alternative approach by breaking down matrix multiplication into parallel multiplication and merging stages. Non-zero elements are multiplied to generate partial product matrices in the parallel stage, and these partial products are then merged to produce the final output. The staged structure of the outer-product algorithm naturally lends itself to parallel computation, particularly advantageous for highly sparse matrices.

3 ARCHITECTURE OVERVIEW

In this section, we introduce the motivations and architecture of FullSparse. The high-level architecture is depicted in Fig 3. FullSparse utilizes a fundamental algorithm based on the outer-product approach.

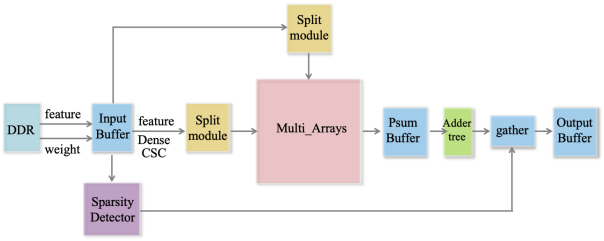

Figure 3: High-level architecture overview of FullSparse.

3.1 Micro Architecture

As shown in Fig 4, the architecture of FullSparse, comprising modules such as Inst_decoder, input_buffer, Split, Multi_array, Psum_buffer, Adder_tree, demonstrates a versatile and optimized design.

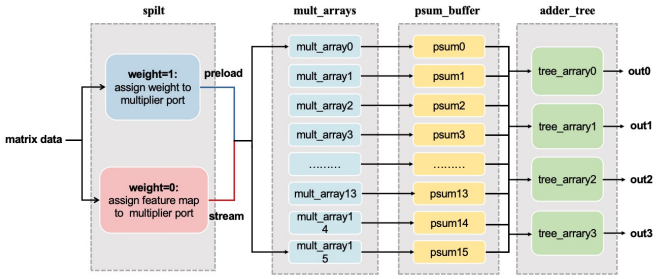

Figure 4: Micro architecture overview of FullSparse.

The Inst_decoder provides custom control instructions for effective weight updates, feature data input mode switching, and diverse output formats. The Input_buffer module cyclically manages matrix data using concatenated SRAM modules to address bit-width limitations. The Split module orchestrates efficient input data flow with Preloading and Multiplication Modes, adapting to different sparsity scenarios.

The Multi_array module, with 4096 pipelined multipliers, efficiently handles pairwise multiplication in both dense and sparse scenarios. The Psum_buffer implements scattering operation in sparse scenarios and serves as a cache in dense scenarios. The Adder_tree module aggregates partial product matrices, while the Adder_array cyclically accumulates results to produce final outputs. Overall, this architecture demonstrates adaptability and efficiency in handling diverse sparsity scenarios.

Figure 5 illustrates the data flow of FullSparse in dense scenarios. When matrix A is dense, the system employs a multiplication array to compute the product of each element in the i-th row of matrix B (64 elements) with each element in the j-th column of matrix A (64 elements). The output of each group of multipliers forms a 64x64 partial product matrix, and within a single cycle, the multiplier array generates 64 such partial product matrices. These matrices are subsequently processed through an adder tree to obtain the final result.

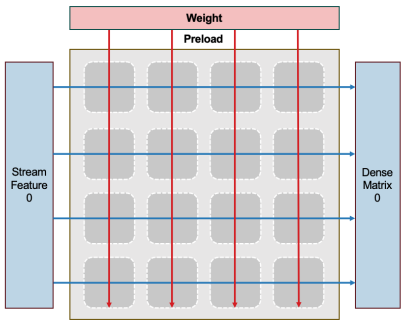

Figure 5: The data flow of FullSparse in dense scenarios FullSparse.

In sparse scenarios, the data flow of FullSparse is illustrated in Figure 6. For matrix A (with a density of non-zeros < 25%), the FullSparse architecture employs a parallel processing strategy. Four feature maps (Stream Feature 0, 1, 2, 3) are simultaneously input, with each column of matrix A restricted to receiving 16 elements per cycle. Excess elements are deferred to the next cycle, with a designated 'finish_flag' indicating when sparse matrix transmission is completed. The system concurrently performs the multiplication of the four instances of matrix A with matrix B, followed by accumulation. The final results are separated into four output matrices. Subsequently, the PSUM module unfolds the sub-matrices, and through subsequent adder tree processing, the final result is obtained.

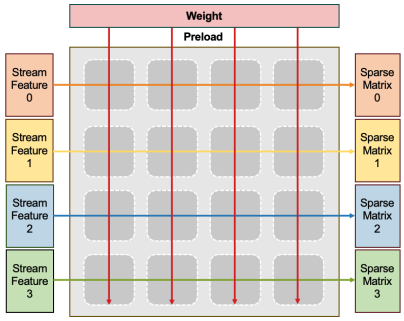

Figure 6: The data flow of FullSparse in sparse scenarios.

3.2 Sparsity Prediction

We present an XGBoost-based system for predicting sparse matrix multiplication outcomes. XGBoost, short for 'eXtreme Gradient Boosting,' is a powerful machine learning algorithm that combines weak predictive models, often decision trees, to create a robust final model. During training, XGBoost optimizes an objective function, balancing prediction error ($l(y_i, \hat{y}_i)$) and regularization term ($\Omega(f_k)$) to control model complexity.

The XGBoost model is meticulously trained to predict outcomes of multiplying two sparse matrices, achieving accuracy rates exceeding 99.9%. Leveraging this online prediction, we proactively schedule the next data batch, enhancing computational efficiency.

4 VERIFICATION AND EVALUATION

In this section, we provide verification and evaluation measurement results of FullSparse.

4.1 Verification Experiment

We completed the RTL design for FullSparse and conducted rigorous functional simulations to validate the accuracy and adherence of the design to specifications. Subsequently, using the DC synthesis tool and a 28nm process library, we performed in-depth evaluations of both power consumption and performance. The results of these evaluations under the 1GHz scenario are succinctly summarized in the following Table 1. We compared FullSparse with PointAcc and Mesorasi, both featuring a 16 * 16 systolic array. ASIC design parameters are detailed in Table 2. Our implementation, while ensuring adaptability to sparsity, demonstrates comparable computational power to the advanced PointAcc.

Module	Area (mm^2)	Power Consumption (mW)
inst_mem	0.1160778176	24.524
inst_decoder	0.000027468	0.0147
input_buffer	2.783731545	261.186
split	0.1113465778	69.924
mult_array	0.880477	2450
psum_buffer	2.636772455	914.912
add_tree	2.727541013	1230
Overall	9.2599	4950.5607

Table 1: Area (mm^2) and Power Consumption.

Chip	Mesorasi	PointAcc	FullSparse
Cores	16×16=256	64×64=4096	64×64=4096
Frequency	1GHz	1 GHz	1 GHz
DRAM	LPDDR3-1600	HBM2	DDR4-2133
Technology	16 nm	40 nm	28 nm
Peak Performance	512 GOPS	8 TOPS	7.8 TOPS
Area (mm^2)	-	15.7	9.2

Table 2: Chip Parameters Comparison

4.2 Performance Evaluation

We simulate the FullSparse design, modeling operation counts and thoroughly assessing its performance. The multi-array design, utilizing a full-pipeline architecture, means that the number of operations per cycle (*operations per cycle*) is equal to the total number of operations. The number of operations per second (*operations per second*) is calculated as *operations per cycle* × *frequency* (f), with f set to 1GHz for this experiment.

In the sparse scenario, we generated 5 sparsity levels spaced evenly between 0 and 0.25. For each sparsity level, we randomly generated four sparse matrices in each iteration and conducted 1000 tests to obtain the average total operation count. To account for cases where a column in the feature exceeds 4 elements, requiring input in the next cycle, the completion time for the multiplication of these four sparse feature matrices with weights is determined by the column in the feature with the maximum number of elements. Peak performance is determined at the number of clocks equals to 1, and average performance considers the average over 1000 tests with varying required number of clocks.

Examining the number of matrix multiplications of dimension 16×16 that can be completed per clock cycle (MMPC) at different densities of non-zeros levels reveals noteworthy insights. Table 3 shows the results. In the dense scenario, one matrix multiplication is fixed per cycle. In the sparse scenario, the optimal scenario envisions parallel execution of the multiplication of four sparse matrices during each cycle. However, in practical scenarios, if a column in any of the four sparse matrices contains more than four non-zero elements, additional cycles are required to finalize the computation. Therefore, at 25% density of non-zeros, the average MMPC is approximately 2, resulting in a doubling of speed compared to the dense scenario. At 5% density of non-zeros, the average MMPC can almost reach 4, achieving a fourfold improvement in performance. This speed improvement stems from the ability of the sparse scenario to capitalize on parallel computing advantages.

Table 3: Performance with Varying Sparsity Levels

Density of Non-zeros (%)	5	10	15	20	25	Full
Peak Computing Power (TOPS)	1.664	3.328	4.8	6.336	7.712	8.192
Average Computing Power (TOPS)	1.626	2.9898	3.2185	3.0453	3.5952	8.192
Designed Peak MMPC	4	4	4	4	4	1
Average MMPC	3.9960	3.5955	2.5341	2.0238	1.9704	1

5 CONCLUSION

This paper presents FullSparse, a novel CNN accelerator designed to handle varying sparsity levels in neural networks. FullSparse optimizes processing techniques to improve energy efficiency across different sparsity scenarios, with adaptable modes for seamless switching. To tackle storage overhead, we introduce an online sparsity prediction system that dynamically adjusts during runtime. The reconfigurable data memory architecture efficiently accommodates data in different sparsity states. Experimental results show that our implementation maintains computational power comparable to advanced designs while being adaptable to sparsity.

REFERENCES

[1] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 1–13.

[2] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. 2016. Cnvlutin: Ineffectual-neuron-free deep neural network computing. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 1–13.

[3] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news* 44, 3 (2016), 367–379.

[4] Letian DENG and Yanru ZHAO. 2023. Deep Learning Deep Learning-Based Semantic Based Semantic Feature Extraction Feature Extraction: A Literature Review and A Literature Review and Future Directions. *ZTE COMMUNICATIONS* 21, 2 (2023), 11–17.

[5] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: Efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News* 44, 3 (2016), 243–254.

[6] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[7] Daiyi LI, Yaofeng TU, Xiangsheng ZHOU, Yangming ZHANG, and Zongmin MA. 2022. End-to-End Chinese Entity Recognition Based on BERT-BiLSTM-ATT-CRF. *ZTE Communications* 20, S1 (2022), 27–35.

[8] Bert Moons and Marian Verhelst. 2016. A 0.3–2.6 TOPS/W precision-scalable processor for real-time large-scale ConvNets. In *2016 IEEE Symposium on VLSI Circuits (VLSI-Circuits)*. IEEE, 1–2.

[9] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH computer architecture news* 45, 2 (2017), 27–40.

[10] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. 2017. SCNN: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH computer architecture news* 45, 2 (2017), 27–40.

[11] Dongjoo Shin, Jinmook Lee, Jinsu Lee, and Hoi-Jun Yoo. 2017. 14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 240–241.

[12] Shijin Zhang, Zidong Du, Lei Zhang, Huiying Lan, Shaoli Liu, Ling Li, Qi Guo, Tianshi Chen, and Yunji Chen. 2016. Cambricon-X: An accelerator for sparse neural networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 1–12. https://doi.org/10.1109/MICRO.2016.7783723

[13] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. 2018. A systematic dnn weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European conference on computer vision (ECCV)*. 184–199.