ZeroTetris: A Spacial Feature Similarity-based Sparse MLP Engine for Neural Volume Rendering

Haochuan Wan
ShanghaiTech University
Shanghai, China
wanhch@shanghaitech.edu.cn

Linjie Ma
ShanghaiTech University
Shanghai, China
malj@shanghaitech.edu.cn

Antong Li
ShanghaiTech University
Shanghai, China
liat2023@shanghaitech.edu.cn

Pingqiang Zhou
ShanghaiTech University
Shanghai, China
zhoupq@shanghaitech.edu.cn

Jingyi Yu*†
ShanghaiTech University
Shanghai, China
yujingyi@shanghaitech.edu.cn

Xin Lou*†
ShanghaiTech University
Shanghai, China
louxin@shanghaitech.edu.cn

ABSTRACT

Neural Volume Rendering (NVR), a novel paradigm for the long-standing problem of photo-realistic rendering of virtual worlds, has developed explosively in the past three years. The unique and substantial computational requirements of NVR pose challenge on deploying NVR to existing dedicated accelerator for neural networks. In this work, we propose ZeroTetris, a spacial feature similarity-based sparse multilayer perceptron (MLP) hardware accelerator for NVR. By leveraging the unique similarity-based sparsity between adjacent sampling points in NVR models, ZeroTetris efficiently bypass the computation of zero activations, thereby enhancing energy efficiency. Evaluation results affirm the effectiveness of the proposed design, showcasing ZeroTetris's superior performance in both area and power efficiency compared to other dedicated sparse matrix multiplication or MLP accelerator designs.

CCS CONCEPTS

• **Hardware → Hardware accelerators**; • **Computer systems organization → Neural networks**.

KEYWORDS

Neural Volume Rendering, Sparse Matrix Multiplication, Hardware Accelerator

ACM Reference Format:
Haochuan Wan, Linjie Ma, Antong Li, Pingqiang Zhou, Jingyi Yu, and Xin Lou. 2024. ZeroTetris: A Spacial Feature Similarity-based Sparse MLP Engine for Neural Volume Rendering. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3649329.3655684

1 INTRODUCTION

Triggered by the phenomenal work of Neural Radiance Fields (NeRF) [13] in 2020, the subsequent three years have witnessed the explosion of Neural Volume Rendering (NVR), a data-driven solution to a persistent challenge in computer graphics—the authentic rendering of virtual worlds. As a revolutionary paradigm, the application of NVR extends across diverse domains, spanning from virtual and augmented reality to 3D reconstruction [5, 6] and autonomous driving [12], delivering exceptional reconstruction and rendering performance.

However, apart from the superior performance, NVR methods also entail substantial computational requirements that do not fit the existing graphics hardware or dedicated neural network accelerators. The original NeRF achieves only 0.03 frames per second (FPS)[13] on the high-end NVIDIA V100 GPU[1], far away from the requirement of practical deployment. While many algorithm-level optimizations have been explored to accelerate NVR, achieving real-time performance, they still hinge on large, power-demanding GPU cards, limiting their applications. Moreover, mapping NVR algorithms to existing hardware accelerators for convolutional neural networks (CNNs) poses a nontrivial challenge due to the use of multilayer perceptrons (MLPs) in NVR instead of convolutional networks [18].

Therefore, efforts have been devoted to designing dedicated hardware for NVR applications. As a pioneering work, [18] introduced a specialized architecture for NeRF, achieving much better energy efficiency than GPU and TPU implementations. Subsequently, several dedicated accelerators tailored for NVR have emerged, incorporating diverse optimizations including efficient sampling strategy as demonstrated in [11], consideration of frame familiarity [7], and implementation of hardware-friendly hash encoding [10]. All these works have reported enhanced rendering speed and superior energy efficiency compared to the original GPU implementation of NeRF.

While existing NVR accelerators have explored various possibilities for acceleration, they missed the unique scene-representation property of NVR methods, i.e., NVR models encode 3D scenes. Specifically, existing designs have yet to harness the similarities between cast rays and spatially adjacent sample points along the rays. In this work, we propose ZeroTetris, a spacial feature similarity-based sparse MLP engine for NVR. In particular, we make the following contributions:

- **Sparsity Characterization:** We conduct a systematic analysis of sparsity characteristics and feature similarities between spatially-adjacent sample points, identifying the intrinsic redundancy in NVR computation.
- **Sparse MLP Engine:** We propose ZeroTetris, a dedicated MLP engine, to leverage the unique similarity-based sparsity in the

*Also with Key Laboratory of Intelligent Perception and Human-Machine Collaboration (ShanghaiTech University), Ministry of Education.
†Corresponding author.

DAC '24, June 23–27, 2024, San Francisco, CA, USA

ACM ISBN 979-8-4007-0601-1/24/06
https://doi.org/10.1145/3649329.3655684

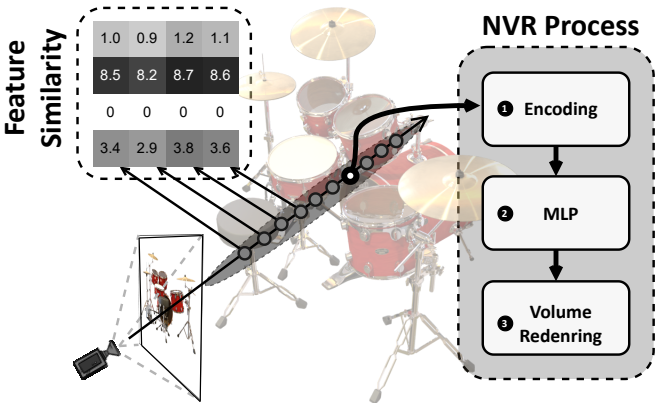

Figure 1: Illustration of the NVR rendering process and feature similarity between spatially adjacent sampling points.

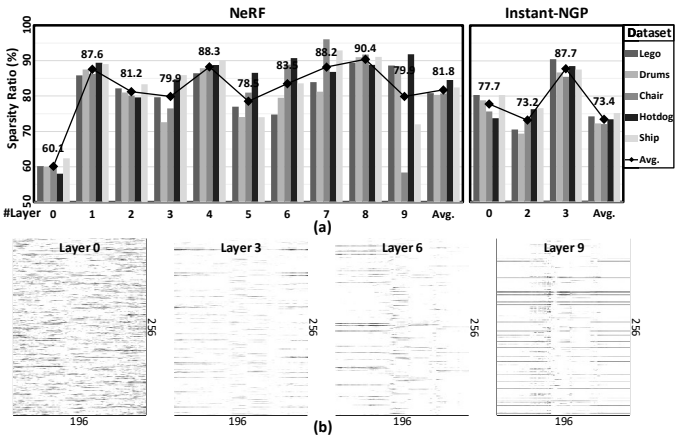

Figure 2: (a) The average sparsity and sparsity at each layer of NeRF and Instant-NGP on the NeRF Synthetic dataset. (b) Sparsity distribution of the output activation matrix for the 1st, 3rd, 6th, and 9th layers, based on 196 samples from NeRF.

the computation of NVR. We also design a multilevel cache architecture for multicore ZeroTetris system to reduce the on-chip memory size and minimize potential data access conflicts.

- **Implementation & Evaluation:** We build a prototype system based on FPGA to validate the proposed ZeroTetris. We also implement ZeroTetris in 28nm CMOS technology and generate the layout for performance evaluation and fabrication. Evaluation results affirm the effectiveness of the proposed design, showcasing ZeroTetris's superior performance in both area and power efficiency.

2 PRELIMINARY & MOTIVATION

2.1 Preliminaries of Neural Volume Rendering

As depicted in Figure 1, the NVR pipeline primarily consists of three steps: ❶**Encoding,** where each sampling point represented by spatial coordinates $\mathbf{x} = (x, y, z)$ and viewing directions $\mathbf{d} = (\theta, \phi)$ is encoded into a feature vector through either fixed (e.g., Fourier-based encoding) or trainable (hash encoding with learnable parameters) functions. ❷**MLP Evaluation,** where the RGB $\mathbf{C} = (r, g, b)$ and opacity σ values are determined by evaluating the MLP for the feature vectors of each sampling point. ❸**Volume Rendering,** where the RGB value of a pixel is finally determined by integrating the colors $\mathbf{C}$ of sampling points based on the corresponding opacity values σ.

In the three steps mentioned above, the significant computation involved in the MLP evaluation step has emerged as the performance bottleneck for NVR algorithms. The original NeRF[13] and the famous Instant-NGP[14] have only 0.4M and 9K weight parameters, respectively, but they require 54T and 289G multiply-accumulate (MAC) operations to render one 800×800 image, respectively. Compared to the well-known ResNet-50 [9], the number of MAC operations of NeRF is four orders of magnitude higher. Therefore, it is crucial to improve the efficiency of MLP evaluation when design custom hardware accelerators for NVR.

2.2 Neural Network & SpGEMM Accelerators

In recent years, many hardware accelerators have emerged, specifically designed for convolutional neural networks (CNNs) or Sparse General Matrix Multiplication (SpGEMM) operations. Early CNN accelerators [3, 4, 16] focus on optimizing the data movement in CNN computations, especially the convolutional operations. They also exploit sparsity existed in CNN weight kernels by offline compressing and encoding of weight parameters, reducing ineffective operations to enhance computational efficiency. However, these accelerators can only exploit static sparsity in weight, as dynamic sparsity in activation necessitates online detection and skipping of zero-values. EIE[8] is able to leverage the sparsity in activations, but may lead to imbalance load among processing elements (PEs). The issue becomes increasingly pronounced with the growing number of PE, making it unsuitable for tasks that demand significantly more computation.

On the other hand, accelerators proposed for SpGEMM tensor algebra [15, 17, 19] excel in accelerating highly sparse and large-sized matrix multiplications, whose dimension can be up to 1.08 billion with just 0.0003% density [15]. To handle these huge sparse matrices and increase utilization of zeros, they develop different formats to encode sparsity information and design specialized module to detect, sort and merge data, which induces large area and power overhead. These designs can exhibit inefficiency when tasked with accelerating NVR algorithms. These algorithms typically feature sparsity levels ranging from 60% to 90% and relatively small matrix dimensions.

3 ZEROTETRIS: THE ALGORITHM

This section introduces the proposed similarity-aware sparse MLP computation algorithm for ZeroTetris. We first conduct an analysis of the sparsity of matrices in the NVR algorithms, revealing the spatial similarity property within activations. Leveraging this property, we propose an outer product-based algorithm to enhance the computational efficiency.

3.1 Spacial Similarity-based Sparsity in NVR

As NVR models implicitly represent 3D scenes, it is nontrivial to exploit the weight sparsity because they may significantly influence the final rendering outcome. However, the use of ReLU in the MLP of the NVR algorithm leads to a significant number of zero values in the activation matrix. Figure 2(a) illustrates the sparse ratio in the activation matrices of each layer for NeRF and Instant-NGP when rendering the NeRF Synthetic dataset [13]. According to our profiling, the average activation sparsity for NeRF and Instant-NGP

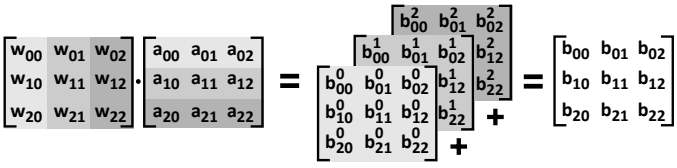

Figure 3: Outer product-based matrix multiplication.

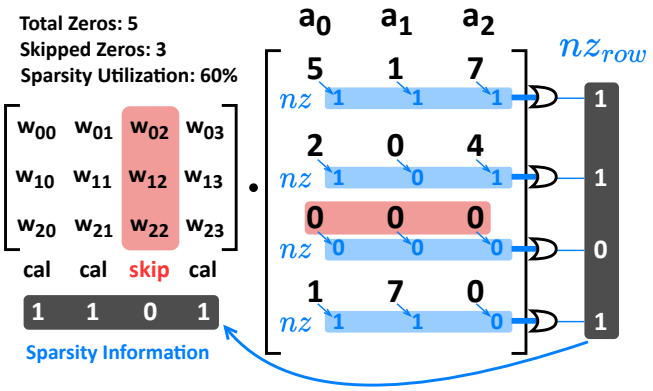

Figure 4: The process of generating sparsity and zero skipping in ZeroTetris, which maximizes the utilization of sparsity.

can reach up to 81.8% and 73.4%, respectively. Moreover, during the rendering process, multiple sampling points are required along each ray as shown Figure 1. Due to the similarity in optical information among adjacent sampling points in the same ray or nearby rays, the corresponding elements in their feature vectors tend to have similar values. This similarity is particularly pronounced after the activation of ReLU, resulting in a similar distribution of zero values in the feature vectors. Figure 2(b) illustrates the sparsity distribution of the output activation matrix for the 1st, 3rd, 6th, and 9th layers, based on 196 samples from NeRF. This similarity-based sparsity tents to make zeros appear in the same row in activation when spatially-adjacent samples querying the MLP in batches.

3.2 Outer Product for Similarity-based Sparsity

Outer product methods [2] transform matrix multiplications into outer productions between columns and rows of two matrices as illustrated in Figure 3. The process can be described as:

$$\mathbf{W} \cdot \mathbf{A} = \sum_i \mathbf{w}_i \otimes \mathbf{a}_i = \sum_i \mathbf{B}_i = \mathbf{B} \tag{1}$$

where $\mathbf{w}_i$ is the ith column of $\mathbf{W}$, $\mathbf{a}_i$ is the ith row of $\mathbf{A}$ and $\mathbf{B}_i$ is the partial result matrix of $\mathbf{B}$. Due to the spatially sparse similarities in activations of NVR (as we have characterized), samples within the same batch may share zeros at the same feature locations, resulting in zeros occupying entire rows. In such cases, zeros in the activation can be efficiently encoded and skipped by rows when utilizing the outer product method for calculation.

As illustrated in Figure 4, each activation row is encoded into nz_{row} based on the presence of non-zero (nz) values. A value of 1 in nz_{row} indicates the presence of nz in that row, while 0 indicates that the entire row consists of zeros. The nz_{row} vector serves as sparsity information, enabling efficient zero utilization. With this information, the fetch of the third column weight can be easily skipped, saving memory access and computation for rows with entirely zeros in the activation. This zero-skipping process mimics

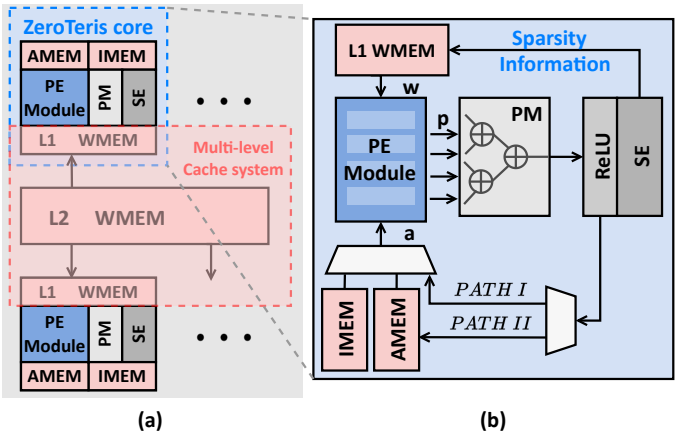

Figure 5: (a) Overview of the proposed ZeroTetris system. (b) Detailed structure of a single ZeroTetris core.

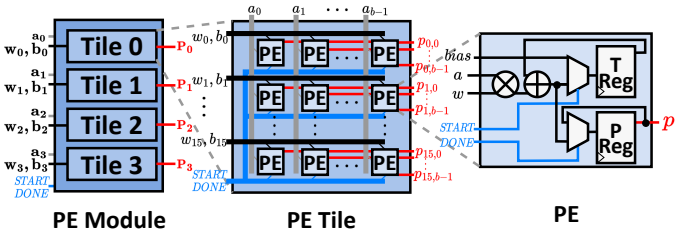

Figure 6: Detailed structure of a PE module.

the mechanics of Tetris, where a row in the activation containing all zeros is "eliminated." However, due to the limitations of this method, zeros that coexist with nz in the same row cannot be effectively skipped. As shown in Figure 4, although the activation has a total of five zeros, only three of them are skipped. A detailed analysis will be presented in §5.2.2.

4 ZEROTETRIS: THE ARCHITECTURE

This section presents the hardware architecture of ZeroTetris. The entirety of ZeroTetris constitutes a multi-core accelerator that shares a common weight memory, as depicted in Figure 5(a). Each core utilizes the outer product-based method to expedite sparse MLP computation. A memory optimization strategy is proposed and implemented between the shared L2 weight memory (WMEM) and the individual L1 WMEM in each core, aimed at enhancing memory utilization and mitigating data request conflicts.

4.1 A Single ZeroTetris Core

As illustrated in Figure 5(b), a single ZeroTetris core primarily consists of a Sparsity Encoder (SE), a Processing Element (PE) module, a Partial result Merger (PM), and three memories, i.e., input memory (IMEM), L1 WMEM and activation memory (AMEM). In each multiplication round, the L1 WMEM selects relevant weight data based on sparsity information and transmits it to the PE module. Activation is sourced either from the IMEM or the result of the previous layer. Once the PE module completes a multiplication round, the partial results are forwarded to the PM, where they are merged to obtain the final result. The SE encodes sparsity information within the ReLU-activated result.

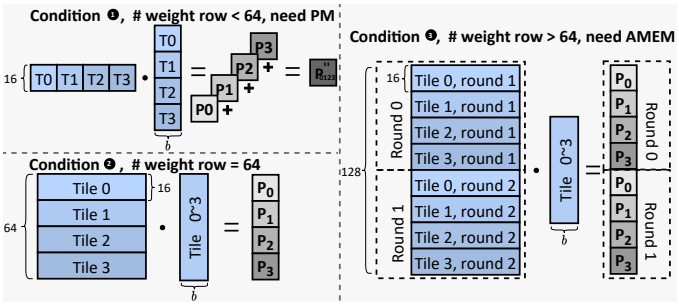

Figure 7: Break the matrix multiplication down into four Tiles for three cases.

4.1.1 The PE Module. As depicted in Figure 6, a PE module comprises four PE Tiles, each housing $16 \times b$ PEs, with b representing the batch size for MLP. At the beginning of a multiplication round, a *START* signal is initiated, and the bias is directly written into the temporary result register (T-Reg). Throughout each cycle of a multiplication round, the module receives b activation data and 64 weight data for the four Tiles, distributed at 16 per Tile. Each PE computes the product of the inputs and accumulates the result in the T-Reg. Upon completing a multiplication round, a *DONE* signal is activated, and the result is stored in the partial result register (P-Reg).

4.1.2 The PM & AMEM. The activation features of hidden layers may have varying dimensions for different NVR algorithms. For instance, the dimension of the output activation in hidden layers is 256 in NeRF, while in Instant-NGP, the output dimension of the 1st, 3rd, and 4th layers is 64, and the 2nd layer is 16. To accommodate these diverse output dimensions in NVR, ZeroTetris introduces PM and AMEM, allowing for adaptation to output dimensions such as 16, 32, 64, 128, 192, and 256. This covers the requirements of most existing NVR algorithms.

When the output activation dimension is less than 64, the partial results computed by the PE module should be accumulated to obtain the final result. The PM retrieves partial results from the P-Reg in the PE module after a round of multiplication and utilizes a two-level adder tree for accumulation. The first-level adder tree combines the partial result matrices $\mathbf{P}0$ and $\mathbf{P}1$ from Tile 0 and Tile 1 to obtain $\mathbf{P'}01$, and similarly combines $\mathbf{P}2$ and $\mathbf{P}3$ from Tile 2 and Tile 3 to obtain $\mathbf{P'}23$. The matrices $\mathbf{P'}01$ and $\mathbf{P'}23$ together form the final result with 32 output dimensions. The second-level adder tree combines the partial result matrices from all four Tiles to obtain $\mathbf{P''}_{0123}$, which represents the final result with 16 output dimensions. When the output activation dimension exceeds 64, the part of final result computed by the PE module need to be cached into AMEM. The AMEM stores data from P-Reg in PE module after a round of multiplication and supports double buffering to ensure sustainable calculation.

Figure 7 illustrates three scenarios explaining the functions of PM and AMEM under different conditions. In **Condition ❶**, where the number of rows of weight, determining the activation output dimensions, is less than 64 (16 in the example of Figure 7), the weight and activation are vertically and horizontally divided into four parts for parallel computation across the four Tiles. This yields four partial results: $\mathbf{P}0$, $\mathbf{P}1$, $\mathbf{P}2$, and $\mathbf{P}3$. The PM adds these partial results to produce the final result $\mathbf{P''}_{0123}$. Since the obtained result is directly needed as input for the next layer's computation, it flows through *PATH I* as shown in Figure 5(b).

In **Condition ❷**, where the number of rows of weight is 64, the result obtained by the four PE Tiles is exactly the final result, and thus, no PM processing is needed. The data also flows in *PATH I* for the same reason as in the first condition. In **Condition ❸**, where the number of rows of weights is more than 64 (128 in the example of Figure 7, the result obtained by the four PE Tiles in each multiplication round is a part of the final result. A total of two rounds is required to obtain the complete result. In this case, the obtained part of the result needs to take *Path II*, as illustrated in Figure 5(b), to enter the AMEM for caching. After the two rounds of computation, it is retrieved from the AMEM and used as input for the next layer's computation.

Algorithm 1 Prefetching and Request Scheduling

```
1: cur_addr = BATCH.next_addr()
2: L1Cache.invalid(cur_addr)
3: prefetch_addr = nz_row.next_prefetch_addr()
4: required_time = nz_row.1s_num(cur_addr, prefetch_addr)
5: L1Cache.send(prefetch_addr, required_time)
6: if nz_row.received() then
7:     L1Cache.clear(loaded_cachelines)
8:     L2Cache.clear(received_memory_request)
9: end if
```

4.2 The Multi-level Cache System

In a practical system, simultaneous data requests from multiple cores to the same memory block in a single cycle can result in data access conflicts, leading to performance degradation. To mitigate conflicts, we introduce a cache system comprising a private L1 cache for each core, responsible for holding and prefetching essential data, and a shared L2 cache that stores all weight data. Algorithm1 illustrates the proposed prefetching and request scheduling algorithm.

4.2.1 Prefetching Algorithm. As detailed in §4.1.2, each column in a weight can be utilized at most once for each batch. Consequently, once these columns are used, the L1 cache can promptly evict them, triggering prefetching (**Line**2). Furthermore, the L1 cache can acquire nz_{row} (§3.2) for each layer, enabling it to skip unnecessary data during prefetching based on nz_{row} (**Line**3). Addresses lacking corresponding nz_{row} at the time of prefetching triggering are marked as needed by the L1 cache. Upon receiving a nz_{row}, both the L1 cache and L2 cache can eliminate superfluous data and prefetching requests, respectively (**Line**6-9).

4.2.2 Memory Request Scheduling Algorithm. With nz_{row}s, the L1 cache can consolidate information on when the data will be utilized in prefetching requests. This capability enables the L2 cache to reschedule memory requests, prioritizing the most imminent ones. The required time can be calculated by tallying the occurrences of 1_s in the nz_{row}s between the currently required address and the prefetched address(**Line**4-5). In cases where nz_{row}s have not been received, the L1 cache treats them as entirely zeros.

5 EVALUATION

5.1 Evaluation Setup

We implement a single-core ZeroTetris with a batch size of 16 in 28nm CMOS technology and further extend it to a multi-core

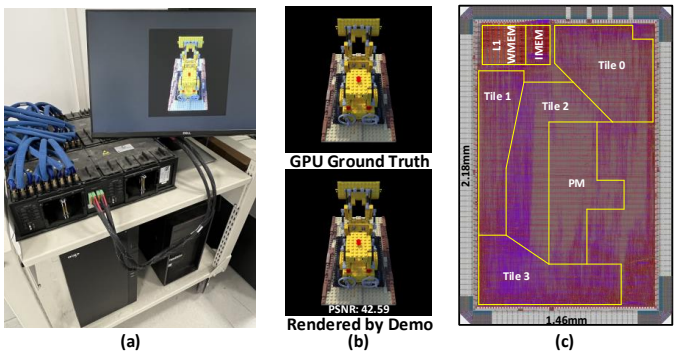

Figure 8: (a) The FPGA-based proof-of-concept prototype system. (b) Image rendered by the FPGA prototype and comparison with the GPU results. (c) Layout of a ZeroTetris core.

Technology	28nm	
Die Area	2.18 mm × 1.46mm (3.18mm^2)	
Power	123.7mW	
Voltage	0.9V	
Frequency	200MHz	
SRAM	L1 WMEM	IMEM
	12.375 KB	3 KB

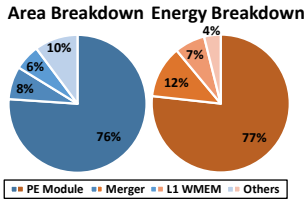

Figure 9: The implementation results of ZeroTetris and corresponding area & power breakdown.

ZeroTetris system with multi-level cache by implementing a cycle-accurate SystemC simulator. We use two well-known NVR algorithms: 1) the original NeRF and 2) a famous follow-up Instant-NGP, together with NeRF synthetic dataset [13] to evaluate the proposed ZeroTetris. To implement NeRF, we include an IMEM of 6 KB, a dual-port L1 WMEM of 16 KB and an AMEM of 16 KB within each core. Multiple cores share a public dual-port L2 WMEM of 960 KB. For Instant-NGP, due to the reduced data size of weights, each core is equipped with its own L1 WMEM of 12.375 KB, eliminating the necessity for a shared L2 WMEM or weight memory optimization. Furthermore, given that all output dimensions are below 64, there is no requirement for AMEM caching.

5.2 Performance and Analysis

5.2.1 Implementation Results. We develop an FPGA-based prototype system as shown in Figure 8(a) to validate the function of the proposed ZeroTetris. We also implement our design in 28nm CMOS process and produce the layout for fabrication as shown in Figure 8(c) for performance evaluation. Power consumption is estimated using PrimePower with real scene data as stimulus. As a common practice, we use Peak Signal-to-Noise Ratio (PSNR) to measure the reconstruction quality. The PSNR between the rendering results of ZeroTetris and the GPU-rendered image is measured at 42.59. This value signifies an exceptionally high reconstruction quality of ZeroTetris. As shown in Figure 8(c), a ZeroTetris core, constrained to operate at 200MHz, occupy a silicon area of 3.18 mm^2 after layout, and consumes 123.7 mW of power. The detailed breakdown of area and power is further illustrated in Figure 9.

5.2.2 Performance Analysis of a Single ZeroTetris Core . As ZeroTetris identifies and skips zeros only when all activations in a given row are zeros, our approach faces limitations in fully exploiting the inherent sparsity within activations. The efficiency of

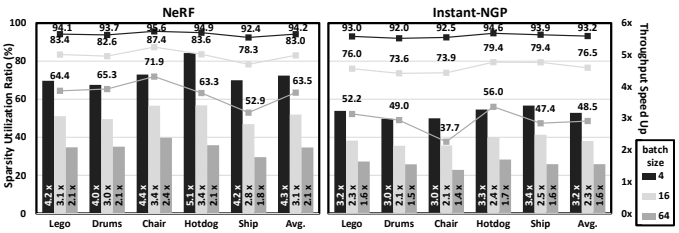

Figure 10: Sparsity utilization ratio and speed up against dense computation method in NeRF and Instant-NGP.

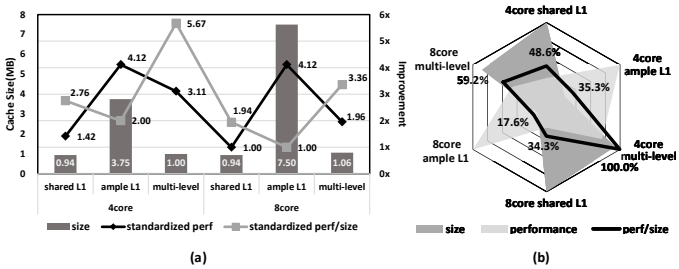

Figure 11: (a) The evaluation results of different cache system. (b) Comparison between cache systems

ZeroTetris in handling zero values is contingent upon the batch size. A decrease in batch size enhances ZeroTetris's ability to discern zero values with finer resolution, leading to a higher likelihood of entire rows being skipped, thereby increasing sparsity utilization.

However, the trade-off exists as reducing the batch size lowers the weight reuse rate. Figure 10 visually depicts the utilization rates for zero values in each layer of NeRF and Instant-NGP under different batch sizes, accompanied by speedup factors in comparison to a dense accelerator. As NeRF and Instant-NGP activations demonstrating 83.0% sparsity and 76.5% zero utilization, ZeroTetris achieves a speedup of 3.1× and 2.3× respectively, when compared to a dense accelerator executing equivalent network computations.

Table 1 presents a comprehensive comparison of ZeroTetris with other existing dedicated accelerators for sparse matrix multiplication and MLP in NVR. In comparison to NeuRex[10], an MLP accelerator designed for NVR, the proposed ZeroTetris exhibits superior area and power efficiency. This is attributed to its ability to effectively bypass the distinctive sparse patterns in the activations. While sparse matrix multiplication accelerators, such as OuterSPACE [15] and SIGMA [17], achieve an impressive sparsity utilization ratio of nearly 100%, their substantial area and power overhead compromises the overall efficiency in terms of area and power. Despite ZeroTetris demonstrating lower zero utilization than other designs, its efficiency lies in leveraging spacial similarity-based sparsity within activations through NVR. By employing a simpler circuit for zero utilization, ZeroTetris incurs smaller overhead in zero-skipping operations. This strategic approach translates into advantages for ZeroTetris in terms of area utilization and energy consumption when compared to alternative designs.

5.2.3 Evaluation of the Multi-Core ZeroTetris System. In crafting a cache system, the pivotal parameters of consideration are area and performance. Consequently, two prevalent types of cache systems emerge in hardware accelerator designs. One adopts a shared

Table 1: Performance Comparison with Other Related Designs

	EIE[8]	OuterSPACE[15]	SIGMA[17]	NeuRex-Server[10]	NeuRex-Edge[10]	ZeroTetris
Applications	DNN	Tensor Algebra	Tensor Algebra	NVR	NVR	NVR
Quantization	Fixed 4bit	Float	Float	-	-	Fixed 11-16bit
Sparsity Utilization Ratio [%]	~100	~100	~100	N.A.	N.A.	83
Process [nm]	28	32	28	28	28	28
Area [mm^2]	63.8	86.74	65.1	14.50	1.66	3.18
Frequency [MHz]	1200	1500	500	1000	1000	200
Power [W]	2.36	23.99	22.33	3.87	0.73	0.124
Power Efficiency [TOPS/W]	1.43	0.18	4.03	8.47	2.81	10.31

single-port L1 cache to minimize area, while the alternative approach allocates ample L1 cache for each core to optimize performance. In contrast, our proposed multi-level cache system excels in maximizing performance per MB.

We implement and test a Multi-core ZeroTetris System incorporating the proposed multi-level cache system using a SystemC-based simulator. Figure 11(a) visually presents the area, standardized performance and standardized performance per MB for various cache system configurations with one batch test. As we can see in Figure 11(b), the multi-level cache system outperforms the other two configurations in both 4-core and 8-core scenarios. Specifically, the multi-level cache system exhibits the best performance in the 4-core configuration, outperforming the other configurations by 17.6% to 59.2% in other scenarios.

6 CONCLUSIONS

This paper introduces ZeroTetris, a hardware design specifically tailored to accelerate sparse MLP operations, leveraging feature similarity-based sparsity inherent in NVR algorithms. Additionally, a complementary multi-level cache for multicore ZeroTetris is also proposed. We build an FPGA-based prototype system to validate the proposed ZeroTetris. We also implement our design in 28nm CMOS technology to evaluate the performance. Operating at 200 MHz, our single core ZeroTetris occupies 3.18 mm^2 silicon area, and consumes 123.7 mW. When compared to the state-of-the-art accelerators for sparse matrix multiplication and MLP computation in NVR, our design demonstrates superior performance in both area and power efficiency.

ACKNOWLEDGMENTS

This work was supported in part by the National Key Research and Development Program of China (No. 2023YFB4404000) and in part by the Central Guided Local Science and Technology Foundation of China (No. YDZX20223100001001).

REFERENCES

[1] 2023. Tesla V100. https://www.nvidia.cn/data-center/v100/. accessed 2023-11-13.
[2] Aydin Buluc and John R. Gilbert. 2008. On the representation and multiplication of hypersparse matrices. In *2008 IEEE International Symposium on Parallel and Distributed Processing*. 1–11. https://doi.org/10.1109/IPDPS.2008.4536313
[3] Yunji Chen, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and Olivier Temam. 2016. DianNao Family: Energy-efficient Hardware Accelerators for Machine Learning. *Communications ACM* 59, 11 (2016), 105–112.
[4] Yu-Hsin Chen, Tushar Krishna, Joel S. Emer, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (2017), 127–138. https://doi.org/10.1109/JSSC.2016.2616357
[5] Nashwan Dawood, R Marasini, and John Dean. 2009. 19 VR–Roadmap: A vision for 2030 in the built environment. *Virtual Futures for Design, Construction and Procurement* (2009), 261.
[6] Francesco Fassi, Alessandro Mandelli, Simone Teruggi, Fabrizio Rechichi, Fausta Fiorillo, and Cristiana Achille. 2016. VR for cultural heritage: A VR-WEB-BIM for the future maintenance of Milan's cathedral. In *Augmented Reality, Virtual Reality, and Computer Graphics: Third International Conference, AVR 2016, Lecce, Italy, June 15-18, 2016. Proceedings, Part II 3*. Springer, 139–157.
[7] Donghyeon Han, Junha Ryu, Sangyeob Kim, Sangjin Kim, Jongjun Park, and Hoi-Jun Yoo. 2023. MetaVRain: A Mobile Neural 3-D Rendering Processor With Bundle-Frame-Familiarity-Based NeRF Acceleration and Hybrid DNN Computing. *IEEE Journal of Solid-State Circuits* (2023).
[8] 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.
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
[10] Junseo Lee, Kwanseok Choi, Jungi Lee, Seokwon Lee, Joonho Whangbo, and Jaewoong Sim. 2023. NeuRex: A Case for Neural Rendering Acceleration. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 1–13.
[11] Chaojian Li, Sixu Li, Yang Zhao, Wenbo Zhu, and Yingyan Lin. 2022. RT-NeRF: Real-Time On-Device Neural Radiance Fields Towards Immersive AR/VR Rendering. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–9.
[12] Jeffrey Yunfan Liu, Yun Chen, Ze Yang, Jingkang Wang, Sivabalan Manivasagam, and Raquel Urtasun. 2023. Real-Time Neural Rasterization for Large Scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 8416–8427.
[13] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. 2021. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *Commun. ACM* 65, 1 (2021), 99–106.
[14] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. 2022. Instant Neural Graphics Primitives with a Multiresolution Hash Encoding. *ACM Transactions on Graphics* 41, 4 (jul 2022), 1–15. https://doi.org/10.1145/3528223.3530127
[15] Subhankar Pal, Jonathan Beaumont, Dong-Hyeon Park, Aporva Amarnath, Siying Feng, Chaitali Chakrabarti, Hun-Seok Kim, David Blaauw, Trevor Mudge, and Ronald Dreslinski. 2018. Outerspace: An outer product based sparse matrix multiplication accelerator. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 724–736.
[16] 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. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 27–40. https://doi.org/10.1145/3079856.3080254
[17] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 58–70. https://doi.org/10.1109/HPCA47549.2020.00015
[18] Chaolin Rao, Huangjie Yu, Haochuan Wan, Jindong Zhou, Yueyang Zheng, Minye Wu, Yu Ma, Anpei Chen, Binzhe Yuan, Pingqiang Zhou, et al. 2022. ICARUS: A Specialized Architecture for Neural Radiance Fields Rendering. *ACM Transactions on Graphics (TOG)* 41, 6 (2022), 1–14.
[19] Nitish Srivastava, Hanchen Jin, Jie Liu, David Albonesi, and Zhiru Zhang. 2020. MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 766–780. https://doi.org/10.1109/MICRO50266.2020.00068