A SIMD-MIMD Acceleration with Access-Execute Decoupling for Generative Adversarial Networks

Amir Yazdanbakhsh
Georgia Institute of Technology
a.yazdanbakhsh@gatech.edu

Kambiz Samadi
Qualcomm
ksamadi@qti.qualcomm.com

Hadi Esmaeilzadeh
University of California-San Diego
hadi@eng.ucsd.edu

Nam Sung Kim
University of Illinois at Urbana-Champaign
nskim@illinois.edu

ABSTRACT

Generative Adversarial Networks (GANs) leverage a new operator, called transposed convolution, that exposes new challenges for hardware acceleration. This operator first inserts zeros within the multidimensional input and then convolves a kernel over this expanded array to add information to the embedded zeros. The inserted zeros lead to underutilization of the compute resources when a traditional convolution accelerator is used. To alleviate the underutilization of the compute resources, we propose X-GAN, the first GAN accelerator design. Evaluations with seven GAN models shows, on average, 3.5× speedup and 3.1× energy savings over EYERISS without compromising the efficiency of traditional convolution at the cost of merely 7.8% area increase. These results suggest that X-GAN is an effective initial step that paves the way for accelerating next generation deep models.

1 INTRODUCTION

While GANs are set to push the frontier across various domains [1–15], there is a lack of hardware accelerators that address their computational needs. This paper sets out to explore this new dimension from the hardware acceleration perspective. Given the abundance of the accelerators for conventional DNNs [16–43], designing an accelerator for GANs will only be attractive if they pose new challenges in architecture design. Studying the structure of emerging GAN models [5–12], we observe that they use a fundamentally new type of mathematical operator in their generative model called *transpose convolution* (TConv). Transposed convolution aims to extrapolate new information from the input feature map. This contrasts with convolution that aims to interpolate the most relevant information from the input. As such, the transposed convolution operator first inserts zeros within the multidimensional input and then convolves a kernel over this expanded input to add information to the inserted zeros. The transposed convolution in GANs fundamentally differs from the operators in the backward pass of training conventional DNNs as these do not insert zeros. Moreover, although there is a convolution stage in the GAN operator, the inserted zeros cause underutilization of the computing resources available compared to if a traditional convolution accelerator were used. The following highlights the sources of these inefficiencies and outlines the contributions of this paper that alleviates these sources of underutilization, making the first GAN accelerator design possible.

(1) **Performing multiply-accumulate on the inserted zeros is inconsequential.** Unlike conventional convolution, the accelerator should skip over the zeros as they constitute almost 60% of all the operands (Figure 1). Skipping the zeros creates irregular dataflow in the following convolution and diminishes data reuse if not handled adequately in the microarchitecture. To address this challenge, we propose a reording of the output computations that allocates computing rows with similar patterns of zeros to adjacent processing engines. This forced adjacency reclaims data reuse across these neighboring computational units.

(2) **Reordering computation is necessary but breaks the SIMD execution model.** The inserted zeroes even with the reordering creates different patterns of computation when sliding the convolution window. As such, the same sequence of operations cannot be repeated across all the processing engines, breaking the full SIMD execution model. Therefore, we propose a MIMD-SIMD accelerator architecture that exploits repeated patterns in the computation to create different microprograms that execute concurrently in SIMD mode. To maximize benefits from both levels of parallelism, we propose an architecture, called X-GAN, that supports interleaving MIMD and SIMD operations at the granularity of a single instruction.

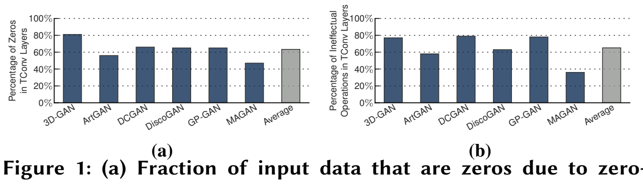

Figure 1: (a) Fraction of input data that are zeros due to zero-insertion in TConv layers and (b) fraction of operations in TConv layers that are ineffectual.

(3) **MIMD is inevitable but its overhead needs to be amortized.** These modifications in the dataflow and the computation order, necessitate irregular accesses to multiple different memory structures while the operations are still the same. That is, the compute part can be SIMD but the access patterns prevent the execution model. For X-GAN, we propose to decouple data accesses from data processing. The decoupling leads to breaking each processing engine to an access micro-engine and an execute micro-engine. The proposed architecture extends the concept of access-execute architecture [44–46] to the finest granularity of computation for each individual operand.

Although X-GAN addresses these challenges to enable efficient execution of transposed convolution operator, it does not impose extra overhead on the execution of traditional convolution, but instead offers the same level of performance and efficiency.

2 ARCHITECTURE DESIGN FOR X-GAN

Figure 2 illustrates the high-level diagram of the X-GAN architecture, which is comprised of a set of identical processing engines (PE) and a memory hierarchy. The PEs are organized in a 2D array and connected through a dedicated network. Each PE is equipped with a local scratchpad register file and a simple arithmetic unit. The memory hierarchy is comprised of an off-chip DRAM and two separate on-chip global buffers, one for data and one for instruction, which are shared across all the PEs. Using a memory hierarchy facilitates the data reuse in the architecture. In this architecture, each PE operates on one row of filter and one row of input and generates one row of partial sum values. The partial sum values are further accumulated together across multiple PEs horizontally to generate the final output value.

Unified SIMD-MIMD architecture. Using a SIMD model for transposed convolution operation leads to resource underutilization. The PEs that perform the computation for sliding windows with fewer number of operations remains idle, wasting computational resources. The simple solution is to replace the SIMD model with a fully MIMD computing model. However, a MIMD execution model requires augmenting each processing engine with an instruction buffer. To prevent the large area overhead of adding an instruction buffer to each PE, we design X-GAN architecture upon this observation that *PEs in the same row perform same operations for a large period of time*. To enable a low-overhead SIMD/MIMD model of

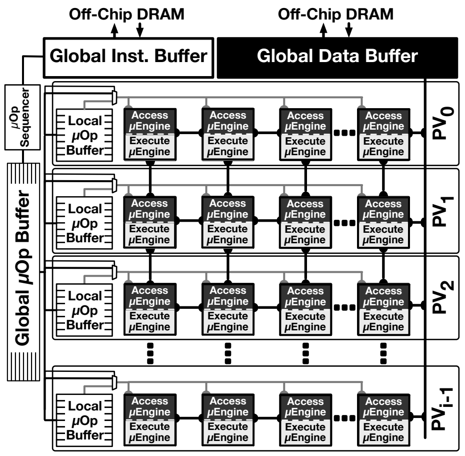

Figure 2: Top-level block diagram of X-GAN architecture.

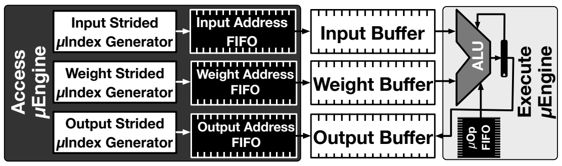

Figure 3: Organization of decoupled Access-Execute architecture.

computing, we introduce a two-level μop buffer (Figure 2). Each horizontal group of PEs, called processing vector (PV), shares a local μop. However, the global μop buffer is shared across all the PVs. The global μop buffer is connected to a global instruction buffer. The accelerator does not execute the instruction from the global instruction buffer. Instead, each instruction is translated to a series of μops in the global μop buffers. The translation from the instruction to μops occurs in the μop sequencer. The accelerator can operate in two modes: SIMD mode and SIMD-MIMD mode. Since all the sliding windows in the convolution have the same number of operations, we use SIMD mode. In the SIMD mode the global μop buffer bypasses the local μop and broadcasts the μop to all the PEs. On the other hand, since the number of operations varies from one sliding window to another in transposed convolution, the accelerator works in hybrid SIMD-MIMD mode. In the hybrid SIMD-MIMD mode, the global μop buffer sends distinct indices to each local μop buffer. Upon receiving the index, each local μop buffer reads a μop and broadcasts it to all the underlying PEs.

Decoupled access-execute μengines. Although the data access patterns in the transposed convolution operation are irregular, they are still structured. Furthermore, the data access patterns are repetitive across the sliding windows. Leveraging this observation, we devise a microarchitecture that decouples data access from from execution. Figure 3 illustrates the organization of our proposed decoupled access-execute architecture. The decoupled access-execute architecture consists of two microarchitectural units, one for address generation (access μengine) and one for performing the operations (execute μengine). The μops of these two units are entirely segregated. However, the access and execute μengines work cooperatively to carry out an operation. The μops for access μengine handle the configuration of address generator units and calculating the memory addresses. The μops for execute μengine *only* specify the type of operation to be performed on the fetched data. As such, the execute μops do *not* need to include any fields for specifying the source/destination operands. Every cycle, the access μengine calculates the addresses for source and destination operands based on its preconfigured parameters. Then, the execute μengine performs the specified operation on the source operands. The result of the operation is stored in the location that is defined by the access μengine. Having two segregated μengines for accessing the data and executing the operations has a paramount benefit of re-using execute μops. Since there is no address field in the execute μop, we can re-use the same execute μop on different data over and over again without the need to change any fields in the μops. Re-using the same μop on different data significantly reduces the size of μop buffers.

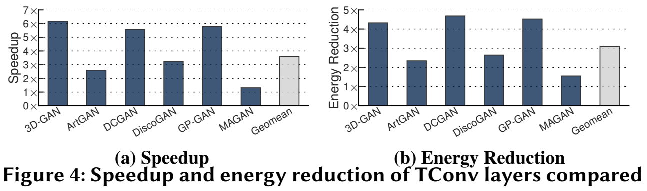

Figure 4: Speedup and energy reduction of TConv layers compared to Eyeriss [19].

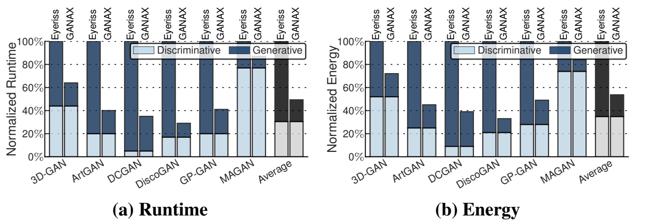

Figure 5: Breakdown of (a) runtime and (b) energy consumption between discriminative and generative models normalized to runtime and energy consumption of Eyeriss, respectively. For each network, the first (second) bar show the normalized value when the application is executed on Eyeriss (X-GAN).

3 EVALUATION AND METHODOLOGY

Experimental setup. We evaluate the performance and energy benefits of X-GAN across a variety of GAN models with our cycle-level simulator and a combination of logic synthesis and detailed energy modeling. For the baseline accelerator, we use Eyeriss [19], one of the most recent and efficient proposals for CNN acceleration. We implement the X-GAN microarchitectural units in Verilog and synthesize using `TSMC 45nm` standard-cell library. To measure the area and read/write access energy of the register files, SRAMs, and local/global buffers, we use CACTI-P [47]. The same frequency (500 MHz) is used for both Eyeriss and X-GAN in all of the experiments. Under this setting, X-GAN architecture has an area overhead of 7.8% compared to Eyeriss.

Experimental results. Figure 4a shows the speedup of the generative models with X-GAN over Eyeriss [19]. On average, X-GAN yields 3.5× speedup improvement over the baseline CNN accelerator (Eyeriss [19]). The GAN models with a larger fraction of inserted zeros in the input data and larger number of ineffectual operations in transposed convolution layers enjoy a higher speedup with X-GAN. Figure 4b shows the energy reductions achieved by X-GAN normalized to Eyeriss [19]. On average, X-GAN effectively reduces the energy consumption by 3.1× over Eyeriss accelerator. Figure 5 shows the normalized runtime and energy breakdown between discriminative and generative models. The first (second) bar shows the normalized runtime (energy) for Eyeriss (X-GAN). As the results show, while X-GAN significantly reduces both the runtime and energy consumption of generative models, it delivers the same level of efficiency for discriminative models. These results show that our proposed architecture is efficient in addressing the main sources of inefficiencies in generative models.

4 CONCLUSION

X-GAN is a unified architecture that brings together both SIMD and MIMD execution models to maximize the efficiency of the accelerator for both generative and discriminative models. To support this mixed mode, each computational unit offers a decoupled micro access-execute paradigm in the finest granularity of its computation micro engines. The evaluation results across a variety of GAN models shows that X-GAN delivers significant performance gains for the generative without sacrificing the execution efficiency of the conventional DNNs.

REFERENCES

[1] Dong Nie, Roger Trullo, Jun Lian, Caroline Petitjean, Su Ruan, Qian Wang, and Dinggang Shen. Medical Image Synthesis with Context-aware Generative Adversarial Networks. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 2017.

[2] Pedro Costa, Adrian Galdran, Maria Ines Meyer, Meindert Niemeijer, Michael Abràmoff, Ana Maria Mendonça, and Aurélio Campilho. End-to-end Adversarial Retinal Image Synthesis. *IEEE Transactions on Medical Imaging*, 2017.

[3] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *NIPS*. 2016.

[4] Arna Ghosh, Biswarup Bhattacharya, and Somnath Basu Roy Chowdhury. SAD-GAN: Synthetic Autonomous Driving using Generative Adversarial Networks. *arXiv preprint arXiv:1611.08788*, 2016.

[5] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *arXiv preprint arXiv:1511.06434*, 2015.

[6] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *NIPS*, 2016.

[7] Ruohan Wang, Antoine Cully, Hyung Jin Chang, and Yiannis Demiris. MAGAN: Margin Adaptation for Generative Adversarial Networks. *CoRR*, 2017.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[9] Dong Nie, Roger Trullo, Caroline Petitjean, Su Ruan, and Dinggang Shen. Medical Image Synthesis with Context-Aware Generative Adversarial Networks. *CoRR*, 2016.

[10] Wei Ren Tan, Chee Seng Chan, Hernan Aguirre, and Kiyoshi Tanaka. Art-GAN: Artwork Synthesis with Conditional Categorial GANs. *arXiv preprint arXiv:1702.03410*, 2017.

[11] Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to Discover Cross-Domain Relations with Generative Adversarial Networks. *CoRR*, 2017.

[12] Huikai Wu, Shuai Zheng, Junge Zhang, and Kaiqi Huang. GP-GAN: Towards Realistic High-Resolution Image Blending. *arXiv preprint arXiv:1703.07195*, 2017.

[13] Li-Chia Yang Yi-Hsuan Yang Hao-Wen Dong, Wen-Yi Hsiao. Musegan: Symbolic-domain music generation and accompaniment with multi-track sequential generative adversarial networks. 2017.

[14] Li-Chia Yang, Szu-Yu Chou, and Yi-Hsuan Yang. MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation using 1D and 2D Conditions. *arXiv preprint arXiv:1703.10847*, 2017.

[15] Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. MuseGAN: Symbolic-domain Music Generation and Accompaniment with Multi-track Sequential Generative Adversarial Networks. 2017.

[16] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. DianNao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. In *ASPLOS*, 2014.

[17] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam. ShiDianNao: Shifting Vision Processing Closer to the Sensor. In *ISCA*, 2015.

[18] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An Energy-efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *JSSC*, 2017.

[19] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. In *ISCA*, 2016.

[20] Mingyu Gao, Jing Pu, Xuan Yang, Mark Horowitz, and Christos Kozyrakis. TETRIS: Scalable and Efficient Neural Network Acceleration with 3D Memory. 2017.

[21] S. Zhang, Z. Du, L. Zhang, H. Lan, S. Liu, L. Li, Q. Guo, T. Chen, and Y. Chen. Cambricon-X: An Accelerator for Sparse Neural Networks. In *MICRO*, 2016.

[22] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks. In *ISCA*, 2017.

[23] Bilel Belhadj, Antoine Joubert, Zheng Li, Rodolphe Héliot, and Olivier Temam. Continuous Real-World Inputs Can Open Up Alternative Accelerator Designs. In *ISCA*, 2013.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *ISCA*, 2016.

[25] Steven K. Esser, Paul A. Merolla, John V. Arthur, Andrew S. Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J. Berg, Jeffrey L. McKinstry, Timothy Melano, Davis R. Barch, Carmelo di Nolfo, Pallab Datta, Arnon Amir, Brian Taba, Myron D. Flickner, and Dharmendra S. Modha. Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing. *CoRR*, 2016.

[26] Schuyler Eldridge, Amos Waterland, Margo Seltzer, Jonathan Appavoo, and Ajay Joshi. Towards General-Purpose Neural Network Computing. In *PACT*, 2015.

[27] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural Acceleration for General-Purpose Approximate Programs. In *MICRO*, 2012.

[28] Amir Yazdanbakhsh, Jongse Park, Hardik Sharma, Pejman Lotfi-Kamran, and Hadi Esmaeilzadeh. Neural Acceleration for GPU Throughput Processors. In *MICRO*, 2015.

[29] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos. Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing. In *ISCA*, 2016.

[30] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A Unified Template-based Framework for Accelerating Statistical Machine Learning. In *HPCA*, 2016.

[31] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh. From High-Level Deep Neural Models to FPGAs. In *MICRO*, 2016.

[32] Thierry Moreau, Mark Wyse, Jacob Nelson, Adrian Sampson, Hadi Esmaeilzadeh, Luis Ceze, and Mark Oskin. SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration. In *HPCA*, 2015.

[33] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun. NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision. In *CVPR Workshops*, 2011.

[34] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *FPGA*, 2015.

[35] Renée St Amant, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, Hadi Esmaeilzadeh, Arjang Hassibi, Luis Ceze, and Doug Burger. General-Purpose Code Acceleration with Limited-Precision Analog Computation. In *ISCA*, 2014.

[36] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. ISAAC: A Convolutional Neural Network Accelerator with In-situ Analog Arithmetic in Crossbars. In *ISCA*, 2016.

[37] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory. In *ISCA*, 2016.

[38] Yufei Ma, Yu Cao, Sarma Vrudhula, and Jae-sun Seo. Optimizing Loop Operation and Dataflow in FPGA Acceleration of Deep Convolutional Neural Networks. In *FPGA*, 2017.

[39] Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding. In *ICLR*, 2016.

[40] Wenyan Lu, Guihai Yan, Jiajun Li, Shijun Gong, Yinhe Han, and Xiaowei Li. FlexFlow: A Flexible Dataflow Accelerator Architecture for Convolutional Neural Networks. In *HPCA*, 2017.

[41] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. PipeLayer: A Pipelined ReRAM-based Accelerator for Deep Learning. In *HPCA*, 2017.

[42] Beayna Grigorian and Glenn Reinman. Accelerating Divergent Applications on SIMD Architectures using Neural Networks. In *ICCD*, 2014.

[43] Xuan Yang, Jing Pu, Blaine Burton Rister, Nikhil Bhagdikar, Stephen Richardson, Shahar Kvatinsky, Jonathan Ragan-Kelley, Ardavan Pedram, and Mark Horowitz. A Systematic Approach to Blocking Convolutional Neural Networks. *CoRR*, 2016.

[44] Decoupled Access/Execute Computer Architectures, author=Smith, James E, booktitle=ACM SIGARCH Computer Architecture News, year=1982,.

[45] Kai Wang and Calvin Lin. Decoupled Affine Computation for SIMT GPUs. In *ISCA*, 2017.

[46] Efficient Data Supply for Hardware Accelerators with Prefetching and Access/Execute Decoupling.

[47] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi. CACTI-P: Architecture-level Modeling for SRAM-based Structures with Advanced Leakage Reduction Techniques. In *ICCAD*, 2011.