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

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

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

Kambiz Samadi  
Qualcomm  
ksamadi@qti.qualcomm.com

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 transposed 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.

1.1 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 reordering of the output computations that allocates computing rows with similar patterns of zeros to adjacent processing engines. This forced adjacency claims data reuse across these neighboring computational units.

1.2 Reordering computation is necessary but breaks the SIMD execution model. The inserted zeros 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.

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 3: Organization of decoupled Access-Execute architecture. 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 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.

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


[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.


