2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID) | 978-1-6654-4087-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/VLSID51830.2021.00061

Design Considerations for Edge Neural Network Accelerators: An Industry Perspective

Arnab Raha, Sang Kyun Kim, Deepak A. Mathaikutty, Guruguhanathan Venkataramanan,
Debabrata Mohapatra, Raymond Sung, Cormac Brick, Gautham N. Chinya
Edge.AI Advanced Architecture Group, Intel Corporation, USA
Email: {arnab.raha, sang.kyun.kim, deepak.a.mathaikutty, guruguhanathan.venkataramanan,
debabrata.mohapatra, raymond.sung, cormac.brick, gautham.n.chinya}@intel.com

***Abstract*—AI acceleration is one of the most actively researched fields in IP and system design. The introduction of specialized AI accelerators in the cloud and at the edge has made it possible to deploy large scaled AI solutions that automate tasks that were previously not possible without AI. The growth of big data and the compute horsepower needed to process this data to provide business intelligence is key for several companies in gaining a competitive edge. AI workloads are both data and compute intensive and improving the efficiency often requires an end-to-end solution. In this perspective paper, we identify key considerations for the design of AI accelerators. The focus of this paper is on Deep Neural Networks (DNN) and how to build efficient deep neural network accelerators through microarchitectural exploration, energy efficient memory hierarchies, flexible dataflow distribution, domain-specific compute optimizations and finally hardware-software co-design techniques. The importance of interconnect topology and the impact of its scaling to the physical design of an AI accelerator is also a key consideration that is described in this paper. In the future, the energy efficiency of these accelerators may rely on approximation computing, compute-in-memory and runtime flexibility for significant improvement.**

I. Introduction

The breakthroughs achieved by neural networks to solve challenging problems in the domain of computer vision, speech recognition, and Natural Language Processing (NLP) has ushered in a new era of demand for high performance compute and domain specific acceleration. The capabilities of machines to work alongside humans and automate tasks has generated a new wave of AI enabled applications that will have a profound impact on society in the coming years. The field of AI is still rapidly evolving and there has been a recent focus on improving the energy efficiency of neural networks at the algorithm level, at the network architecture level through network search [1] and through hardware implementations that can take advantage of the domain specific nature of work. Hardware acceleration, in particular, has been focused on employing lower precision data types, exploiting sparsity and maximizing reuse to improve the efficiency of tensor operations in a neural network. The vast majority of existing AI-enabled applications are still run on CPUs and GPUs due to the ease of programming and availability of high level frameworks that make it easier to experiment with network parameters and deploy these solutions from the cloud to the edge. However, these solutions are not as energy-efficient, nor do they match the throughput of domain specific accelerators. As networks become more complex, the energy required for doing training and inference has resulted in a noticeable shift towards adopting specialized accelerators to meet strict latency and energy constraints that are prevalent in both edge and cloud deployments. In this paper, we will focus on design considerations of custom edge inference accelerators. We will also examine how to scale the throughput and power in an era where the architectural co-design of hardware and software is key to optimize the entire system. There are fundamental limits to scaling the throughput of custom AI accelerators and for the industry to succeed, the problem needs to be attacked from multiple angles to create scalable AI solutions.

In 2012, AlexNet [2], a DNN that uses supervised learning for image classification, rekindled interest in AI and domain specific accelerators for AI. Following that seminal work, researchers have proposed several alternate network topologies to solve the image classification problem with improved accuracy and fewer parameters. Two key ideas include:

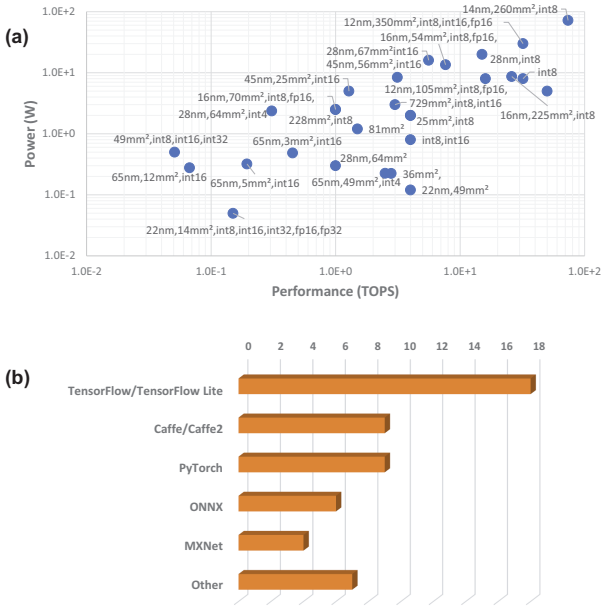

Fig. 1: (a) Edge AI accelerator competitive landscape. Plotted with public data. Baseline reference [6] (b) Popular AI frameworks supported by 20 commercial accelerators.

residual networks [3], which aim to build deeper networks and mobilenets [4], which popularized depth-wise separable convolutions. These have greatly improved the accuracy and efficiency of neural networks. The total amount of compute required for training has reduced over time compared to the original AlexNet. As described in the Open AI blog [5] there has been a 44× improvement in efficiency that can be attributed to algorithmic advances with new network topologies and improved training techniques without having a significant decrease in accuracy. These improvements along with transfer learning capabilities that allows for rapid specialization has made it easier to deploy neural networks for computer vision problems.

Advances in deep neural network algorithms have led to a plethora of research and commercial solutions for the architecture of domain specific accelerators and hardware/software co-design. Fig. 1(a) is a visual representation of the competitive landscape for the latest edge AI accelerators. The edge accelerators represent a wide range of power and performance operating points due to the many different use cases that need to be supported. Software support is essential for the performance of the hardware. AI workloads are expressed using frameworks in high-level languages such as Python or C++. As can be seen in Fig. 1(b), most commercial accelerators support at least one of the popular AI frameworks via their Software Development Kits (SDKs). How effectively the software maps workloads onto the accelerator hardware determines the compute efficiency for a neural network, and compilers (TVM, LLVM) and Intermediate Representations (IR) such as MLIR can help bridge the software-hardware gap using various optimization techniques. However, details

2380-6923/21/$31.00 ©2021 IEEE
DOI 10.1109/VLSID51830.2021.00061

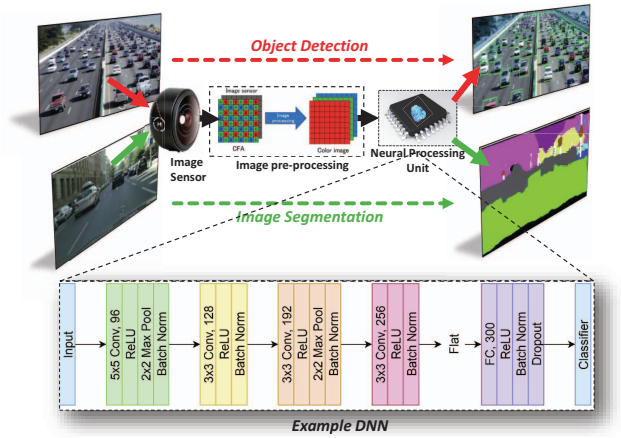

Fig. 2: DNN accelerator running a Neural Network

on software optimizations is beyond the scope of this paper.

TOPS (Tera Operations Per Second), TOPS/Watt, and TOPS/mm^2 are commonly used metrics to compare the performance and power efficiency of DNN accelerators. However, these metrics are limited as they fail to capture how well utilized the compute engines are, the amount of data movement required to perform the compute and the bandwidth constraints on the memory hierarchy. All of these effects, when combined, will dictate how the accelerator performs on real world workloads. As a case study, one accelerator in Fig. 1(a) reports a peak of 32 TOPS based on the computational resources available. However, based on the published results from the vendor, the achievable throughput for ResNet-50 with batch size of 1 is much lower since only a fraction (8% - 9%) of the compute hardware is being utilized. The published TOPS/Watt for an accelerator can also be lower in real-world applications as the system-level power takes into account the power required to move the data across the entire memory hierarchy. Consequently, it is important to examine data movement of the system across all levels of the memory hierarchy as opposed to the energy expended for compute operations alone. The amount of energy spent to move data across these levels often far outweighs the compute energy [7]. Thus, the memory hierarchy and associated data movement costs are important in the design of AI accelerators. A domain specific neural network accelerator can store data closer to compute to reduce data movement and take advantage of domain specific nature of compute as tensor operations are often sparse and the data can be quantized to fewer bits with negligible loss of accuracy. Sparsity, in particular, reduces the amount of data to be stored and transferred and can be exploited to accelerate the compute by skipping zeros [8], [9]. It is important to note that techniques to prune networks also have drawbacks [10] and might not be scalable. It is important to understand the implications of these design choices while considering the targeted application.

This paper identifies key design considerations for DNN edge accelerators and provides insights for reaching the required level of performance and efficiency in a power constrained computing environment. Section II discusses various design approaches and techniques in recent accelerators and why those design decisions were made. Physical design challenges as they relate to timing, layout and technology scaling are discussed in Section III. We conclude by providing some emerging trends in DNN accelerators in Section IV.

II. Design Considerations For DNN accelerators

Neural networks are increasingly used in a multitude of applications ranging from image segmentation, object detection, to speech recognition and natural language processing. Some of the the most popular neural networks are:

- ConvNet: Convolutional Neural Networks (CNNs) are the most popular and widely used neural network for image processing tasks. They rose to prominence after being selected as the most accurate and efficient algorithm for performing image recognition and object detection. A ConvNet model is characterized by a graph where the nodes represent convolution operations using sets of filters obtained through network training. The convolutions are followed by a non-linear activation function, such as a ReLU layer, and sometimes a pooling layer for down-sampling.
  - GAN: Generative Adversarial Networks (GANs) are generative models that use a pair of neural networks for training: a generator to generate (fake) examples and a discriminator to distinguish between real and (fake) generated examples. As most major advancements in GANs were made in computer vision, CNNs are typically used as the generator and discriminator.
- LSTM: Long Short-Term Memory (LSTM) networks are mainly suited for classifying time-series data and are used in applications such as handwriting and speech recognition. LSTM is a Recurrent Neural Network (RNN) architecture with feedback connections. They are characterized by matrix-vector products followed by non-linear activation functions like sigmoid and tanh.
- Transformer: Transformers are a recent development in the field of deep learning and have already gained popularity NLP. A Transformer model is characterized by multiple parallel and scaled dot-product attention units with normalization and softmax layers.
- GNN: Graph Neural Networks (GNNs) provide the learning framework for irregular graphs and gained wide adoption where graph processing is important such as social networking or molecule structures. The goal is for every node to learn the relational nature or property of its surroundings (or the entire graph). GNNs are also computed using matrix-vector products.

As seen above, different types of AI workloads require different network architectures with each one being characterized by different operations performed on tensors (multi-dimensional arrays) and data sets. While there are several types of DNN networks, almost all of them involve convolutions and matrix multiplications [11]. Owing to the ubiquity of CNNs in modern vision applications, which comprise the largest category of applications that run on the edge, we will use CNNs to illustrate design considerations for DNN accelerators. Fig. 2 shows a typical DNN accelerator running one or more of the aforementioned networks. The design principles for CNNs are generic in nature and are applicable to other types of neural networks as well.

Note that within a DNN, the convolutional layers dominate the total number of operations executed, the total time spent, and the total energy expended. Hence, most DNN accelerators attempt to accelerate the convolutional layers. CNNs operate on multi-dimensional arrays called tensors for the input activations and the filter data. Consider an image, represented as a tensor I of height H, width W, containing C input channels, thus having dimensions $H*W*C$. A *Channel* refers to a unique component of an image. A typical RGB image has 3 channels - red, green and blue. The squared filter F which is used to compute the convolution product is of dimension f, and has C input channels and K output channels, total size of $f*f*C*K$. For a 2D feature map with a row index x, a column index y, and an output channel index z, convolution operations to generate an output feature map of height O_y width O_x and K number of output channels can be represented mathematically by:

$$Conv(I,F)_{x,y,z} = \sum_{i=1}^{f}\sum_{j=1}^{f}\sum_{c=1}^{C} F_{i,j,c,z} I_{x+i-1,y+j-1,c}$$
$$(1 \le x \le O_x, 1 \le y \le O_y, 1 \le z \le K)$$

As evident from the above formula, the key computational kernel of a convolution operation is the dot product between two vectors, which consists of a series of Multiply-and-Accumulate (MAC) operations. Even for the seemingly simple task of classifying an image, a DNN requires several billions of MAC operations. Thus, DNN workloads are usually executed using massively parallel MAC arrays either inside of a GPU, where the compute is performed by 1000s of threads that

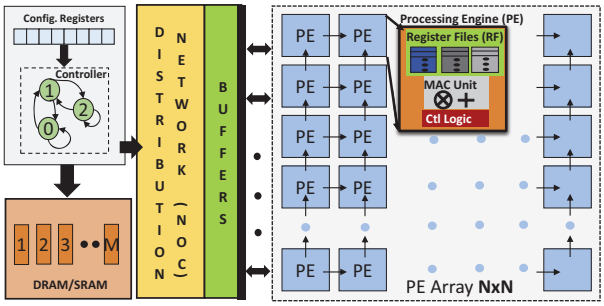

Fig. 3: Typical Edge DNN accelerator microarchitecture

are scheduled to run on a large number of tensor cores that support a matrix-multiply instruction as a primitive or a specialized DNN accelerator. For power constrained edge platforms, energy-efficient DNN accelerators are usually the compute platform of choice.

Edge accelerators mainly comprise of an array of processing elements (PE), capable of doing multiply MAC operations. To exploit the data parallelism in DNNs, the total number of MACs as well as the total number of PEs is typically large, resulting in a parallel compute array structure as shown in Fig. 3. These parallel PE arrays are interconnected by a Network-on-Chip (NoC), which distributes the data needed for the compute and reads the computation results back to memory. To meet the target performance constraints for particular use cases, a majority of the edge DNN accelerators will consider Power, Performance, and Area (PPA) metrics. In addition to the fixed function MAC arrays, most accelerators also consist of programmable DSP/vector cores that are tightly coupled which allows these cores to work in parallel executing certain tensor operations that cannot be mapped to the PE array. This helps in minimizing the total data moved in and out of the subsystem when running the entire network.

A. Compute microarchitecture

As the convolution layers dominate the CNN computation time, some accelerators adopt systolic arrays that are highly optimized for matrix multiplication such as the Google Edge TPU [12]. A systolic array is a 2D grid of MAC units where data flows from top to bottom and left to right providing a substantial amount of data reuse based on the dimension of the 2D grid. Due to the regular structure, systolic arrays are relatively simple to build. The disadvantage of this approach is that it works well only for dense operations and sparsity cannot be easily exploited for acceleration. Furthermore, the matrix dimensions need to be multiple of the grid size to utilize all the compute units. Although several different approaches are proposed to address these deficiencies in [13], they are not widely adopted by the industry.

An alternative microarchitecture, as found in [9], builds a spatial array of PEs as shown in Fig. 3, where each PE has local storage and control logic to process the data. Compared to systolic arrays, PE-based microarchitecture allows a more flexible workload distribution to the MAC resources and makes it easier to exploit sparsity. However, the logic overhead of the local control within the PE adds area and complexity to the design.

Advanced accelerators from the academia use compute in memory [14] to alleviate the memory bottleneck leading to unprecedented amounts of energy savings; however, this is yet to be adopted in any commercially available accelerators due to the fact that a fixed sized storage constraint limits the networks that can be run efficiently as all weights and intermediate activations need to fit within the available fixed amount of memory.

B. Microarchitecture performance

The die area allocated for the compute engines determines the peak TOPS. In general, the performance as measured by TOPS, can be improved through higher operating frequencies. This is enabled through i) a local memory hierarchy to enable compute closer to memory with shorter wires and data reuse, ii) a highly pipelined microarchitecture where the critical path is reduced to that of only a single multiplier in the MAC unit, iii) maximizing the overlap of the various phases of the accelerator operation for maximum performance.

TABLE I: Important parameters for DNN accelerators

Parameter	Objective
Operations per inference	Lower
Operations per cycle	Higher
Clock cycles per second	Higher
Number of PEs	Higher
Number of active PEs	Higher
Effectual operations out of total operations	Higher
Unexploited ineffectual operations per cycle	Lower

The peak achievable TOPS is usually determined by the memory hierarchy and data distribution network to and from the compute engine. This can be enabled through i) sufficient bandwidth allocation for feeding all the PEs in parallel based on the internal reuse factor, and, ii) ensuring high PE utilization and less idle time. Table I shows the relevant performance parameters that need to be considered.

C. Importance of memory hierarchy for energy

DNN inference requires the retrieval of millions of weights and the execution of hundreds of millions to billions of MAC operations to generate the output for a single input. However, in spite of the many computations required, most of the energy is spent in accessing the large amount of data usually corresponding to the retrieval of weights and intermediate activations. Fig. 5 shows the energy contribution of each atomic operation that occurs within a DNN accelerator [7]. It is necessary for ML accelerators to have multiple levels of memory hierarchy to maximize operand reuse at multiple levels as shown in Fig. 4. The faster but smaller capacity memories are kept nearer to the PEs, while the slower and the larger ones are kept farther from the PEs. The memory size also determines the amount of energy spent to access the memory. The idea behind reuse is to frequently access (both temporally and spatially) the smaller and cheaper local memories (RF) and reduce accesses to the larger and more expensive levels of memory (SRAM/DRAM) (Fig. 4). An example of RF integration within the PE is shown in Fig. 3. Here the RF's directly feed the MAC units thus encouraging higher local reuse. For DNN accelerators, there are 3 classes of data reuse [15]: i) convolutional reuse, ii) input feature map reuse, and iii) filter reuse.

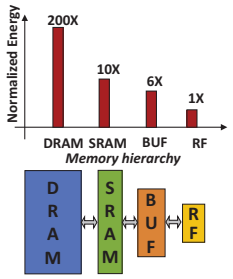

Fig. 4: Energy cost

The size of the RF also dictates the energy to access the structure. There are tradeoffs involved in area *vs.* energy efficiency as increasing local storage closer to the PE will have diminishing returns once sufficient reuse levels are achieved within the PE. Thus, from an energy-efficiency point of view, there exists an optimal RF size for DNN accelerators [16]. Since the memory access patterns for activations and weights in DNNs are known upfront, a tightly-coupled scratchpad memory serving as an intermediate buffer between DRAM and RF is preferred over a highly complex multi-level cache. Caches incur high energy and area overhead due to the inclusion of tags [17].

D. Fixed vs. flexible dataflow

The input tensor dimensions change from one convolution layer to another. Hence, it is important to build flexible hardware accelerators that can process frequently evolving network layers of arbitrary dimensions. Mapping of arbitrary tensor dimensions to a PE array with a fixed tensor mapping pattern can decrease the utilization of the array. As discussed above, reducing data movement, maximizing data reuse from local memory and improving resource utilization are keys to improving performance and energy efficiency. Many existing accelerators for DNN (*e.g.*, Eyeriss [8], [9], TPU [12], SCNN [18])

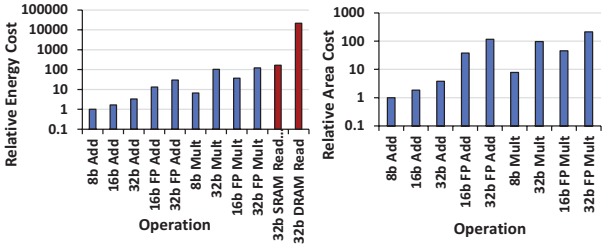

Fig. 5: Energy and area cost of different operations in 45 nm [7]

implement a custom memory hierarchy and a fixed dataflow. This dictates the order in which the tensors for activations and weights are moved into the processing units and the work assigned to each PE. A fixed dataflow constrains the types of data movement across the memory hierarchy and also limits the degree of reuse within the processing units. The data movement and order of reuse determines the energy consumed for each layer. Prior work [19], [20] have attempted to characterize the energy efficiency of DNNs through building analytical models. These emphasize the need to provide the flexibility to schedule tensors of different dimensions by maximizing reuse from the innermost memory hierarchy, where energy cost per unit of data moved is smallest. Vision networks (ResNET, YOLO, VGG) have tens to hundreds of layers, where each layer prefers different dataflows for achieving optimal energy efficiency; fixed-dataflows are only able to provide optimal data reuse and resource utilization for a limited set of the DNN layers. This limits the overall energy efficiency of the network.

Other prominent fixed dataflow accelerators, some of which are meant for use in the cloud as opposed to the edge, include NeuFlow [21] and ISAAC [14], which implement weight stationary schedules, ShiDianNao [22], which implements an output stationary schedule, and MIT's Eyeriss [8], which implements a row stationary schedule. One of the hindrances to flexibility in these accelerators stems from their tensor data distribution hardware that can only address on-die storage in a fixed pattern. This only allows transferring tensor data from the SRAM banks to the PE arrays and storing data back to the SRAMs in a fixed manner.

To implement a flexible dataflow accelerator, it is important to design a dataflow-aware tensor distribution unit that uses layer-specific optimal schedules and dataflow information to distribute the data accordingly. In addition, the accelerator should have implicit support for flexible mapping and execution of this data within each PE. Microsoft's Project Brainwave [23] is an example of a configurable dataflow architecture that uses a vector-vector operation as a primitive. This design takes advantage of the distributed high bandwidth of local memory and a configurable dataflow to maximize the utilization and reduce the latency for inference. Towards that end, there has been a growing trend towards design of flexible accelerators that can support a multitude of dataflows or workloads [8].

E. Domain specific optimization techniques

Energy savings in DNN accelerators is achieved through the reduction of the storage and compute requirements. To this end, three neural network specific techniques for achieving energy-efficient DNN execution are: *i) Quantization, ii) Pruning,* and, *iii) Compression*.

i) *Quantization* refers to the process of mapping a large number of possible values to a smaller set of values using reduced precision for representation. Usually, quantization for DNNs is achieved by converting floating point operands to lower precision fixed-point operands such as INT8, INT4, or INT2, or even ternary and binary values in extreme cases. The impact of quantization can be significant both in terms of area and energy savings. Due to a drastic reduction in the operand bits, the logic complexity for performing MAC operations as well as the storage required for both activations and weights are reduced significantly. This can lead to significant performance improvements due to higher operating frequency and ability to fit more PEs in a given area, and also to energy reduction due to fewer memory accesses and computations.

ii) *Pruning* refers to the process of introducing sparsity by zeroing out weights during training without significantly affecting the accuracy of the network. The two main types of network pruning are: *a) channel pruning* which can be executed on any hardware [24], and *b) weight pruning* which can be leveraged only through specialized hardware [25]. Hybrid pruning [26] is a recent technique that balances these two techniques to zero out more values with lesser impact on accuracy.

iii) *Compression* is another artifact of the existence of sparsity in activations and weights. Zero Valued Compression (ZVC) and Run-Length Encoding (RLE) are lossless compression techniques used by DNN accelerators for reducing memory storage and memory access energy. Furthermore, compression can lead to less memory traffic across the entire hierarchy resulting in higher accelerator throughput.

Despite the obvious benefits of quantization and pruning, some adverse effects on network robustness have been shown. Excessively pruned networks are more vulnerable to attacks and some layers are more sensitive to quantization/pruning than others [10].

Designing for Sparsity: Exploiting sparsity provides opportunities for data compression, which reduces the data footprint of the tensors thereby reducing storage requirements and bandwidth. In addition, it saves energy and improves performance by reducing the number of ineffectual MAC computations done by the PE. However, the cost of exploiting sparsity is the additional hardware that is needed to identify and skip these zero operations within the PE. This adds area and can increase the critical path to feed data if it is not carefully designed. Furthermore, it also lowers the utilization of the active PEs that do not need to compute the zeros and have become idle due to the allocated work imbalance. One example of a fully-sparse architecture is SCNN [18]. Given the optimization techniques mentioned above are also used to introduce sparsity into the tensor data, the non-uniformity of the sparsity has a direct impact on the hardware design of the PE array. The non-uniformity results in PEs operating at different rates, where the slowest PE (operating on low sparsity data) blocks the rate at which the faster PEs (operating on highly sparse data) can proceed. This dependency between the PEs due to the varying sparsity levels introduces inefficiencies into the data distribution network and therefore requires co-designed optimizations between exploiting sparsity versus efficient data distribution.

Activation sparsity mainly arises due to the ReLU activation function. Sparsity-enabled DNN accelerators are designed to exploit either fine granularity sparsity (for each activation/weight) or block-level sparsity (for a group of activations/weights). In addition, sparsity support can also be one-sided or two-sided. Due to the additional cost of sparsity, accelerators can be designed for semi-sparse support where one-sided sparsity either in the feature maps or the filters can be exploited. An example of this is Cnvlutin [27]. Some accelerators only support sparsity for limited types of layers such as fully-connected layers in EIE [28] and convolution layers for SCNN [18]. A comparison of some of these popular architectures are found in Table II. This table compares how sparsity is handled, whether compression saves storage and bandwidth, and whether they incur any loss of accuracy. Commercial AI accelerators such as NVIDIA's A100 Tensor Core [29] and Intel's KeemBay (KMB) [30] also exploit sparsity to improve efficiency of their engine. The A100 exploits structured one-sided fine-grained sparsity at the block-level whereas Intel KMB [30] exploits two-sided fine granular sparsity. Block based sparsity which is supported in NVIDIA's A100 tensor core is simpler to implement, but only improves performance with a max speedup of 2× and forces a fixed number of zeroes to be present within a block which may reduce DNN accuracy. The speedup that can be obtained via sparsity is much larger as later layers in the networks often contain more than 50% of zeroes which cannot be exploited with block sparsity.

F. HW-SW codesign optimization techniques:

An increasingly prevalent trend is to develop customized DNNs that run on customized DNN accelerators for increased efficiency.

TABLE II: Compare and Contrast Sparsity-enabled Accelerators

Architectures	Sparsity	Save Storage & BW	Accuracy Loss
Cnvlutin [27]	One-sided	No	No
Cambricon-S [31]	One-sided	No	Yes
SCNN [18]	Two-sided	Yes	No
EIE [28]	One-sided	Yes	No
Eyeriss v2 [9]	Two-sided	Yes	No
SparseNN [32]	Two-sided	Yes	Yes
Nvidia A100 [29]	One-sided	Yes	No
Intel Keembay [30]	Two-sided	Yes	No

For example, DNNs can be constructed to support weights that are only powers of 2. This enables the construction of a specialized DNN accelerator whose multiplier unit can be implemented using just shift operations [33]. Hardware built with shifters *vs.* multipliers can lead to large area and energy savings but usually come the expense of accuracy. Furthermore, one can use lower precision datatypes that can be exploited through specialized hardware. One such example is Microsoft's Project Brainwave that uses a soft Neural Processing Unit (NPU) programmed on a high-performance Field-Programmable Gate Array (FPGA) to accelerate inference using custom precision. Another significant work that explores the co-design of accelerators with a tensor compiler stack is the Versatile Tensor Accelerator (VTA), an open, generic, and customizable deep learning accelerator with a complete TVM-based compiler stack [34]. TVM and VTA together form an end-to-end hardware-software deep learning system stack that enables end-to-end system level optimizations.

G. Interconnect technology

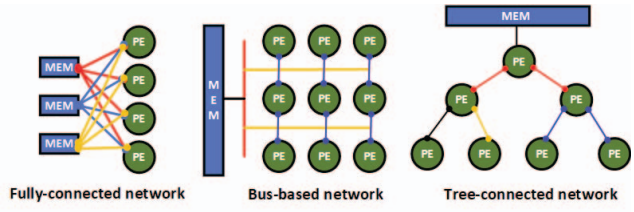

Fig. 6: Typical network topologies seen in DNN accelerators

Interconnect topology is one of the key considerations that need to be addressed to design a scalable high performance DNN accelerator. A fully connected all-to-all interconnect network is extremely inefficient from an area and power perspective when building DNNs with larger than 100 PEs [8]. Therefore, almost all accelerators use an interconnect in the form of a hierarchy of buses and/or trees to connect their cluster of PEs to the on-chip memory as shown in Figure 6. The NoC dictates the topology that determines the layout of PEs and memories and their interconnection. One of the main factors that determine the ideal topology is the distribution pattern support by the mapped dataflow. Given that the goal is to distribute activation and weights to the PEs in an efficient manner, there are three possible distribution patterns that can be supported based on the level of reuse: (i) unicast, (ii) multicast, and (iii) broadcast as shown in Figure 7.

The NoC is also crucial to address scalability, which allows the PE array to be scaled up to achieve higher performance by increasing the amount of resources (number of PEs and on-chip storage). The interconnect also needs to scale to meet the higher bandwidth demands as the PE scales. In practice, this is often difficult due to long-distance wiring, physical implementation challenges and higher latencies associated with data transfers. Hence the following are capabilities an ideal DNN NoC must have :

1. *Wide datapath support:* Allowing large data transfer sizes for increasing the bandwidth of the connection between the PE and memory. This ensures any PE can be fed with the required bandwidth and avoid data starvation due to limited bandwidth.
2. *Physical Design Aware:* Designing the network topology to be aware of the on-chip components and their layout. Relying on modularized design to reduce the number of components connected to a network and being able to compose these smaller networks. This ensures fewer physical implementation challenges and a smoother timing convergence.
3. *Reuse-driven distribution:* The amount of reuse within the engine is increased by distributing one data to many PEs simultaneously through multicast or broadcast style communication through the interconnect. This in turn reduces the off-chip memory accesses and improves computational flexibility. A lower reuse implies higher data bandwidth demand that need to be delivered by the NoC.
4. *Low Implementation cost:* Given the nature of the activation and weight distribution, there is no need for complex interface protocol such as AXI to deliver data to the PE. An interface with minimal implementation overhead will reduce the overall overheads in moving data and reduce design and validation time.

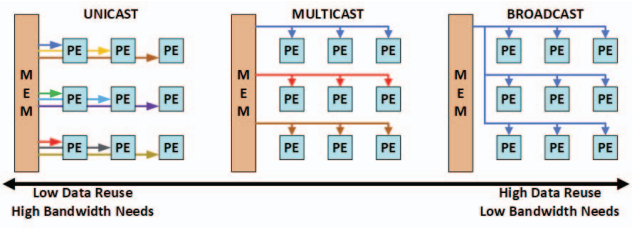

Fig. 7: Data distribution patterns through NoC

The commercially available FlexNoC®AI package [35] from Arteris is one such interconnect technology that allows creating scalable physically aware topologies such as rings, meshes or tori for building an AI accelerator. The Eyeriss v2 [9] is an example accelerator that supports the bandwidth requirements of their engine through a hierarchical mesh NoC to connect the PEs to the global buffers. The NoC also supports unicast distribution of data to the PEs when the data reuse is low and multicast or broadcast distribution to exploit spatial data reuse. The DNN accelerator MAERI [36] uses a binary-tree based NoC topology capable of multicast as well reconfigurable sizing of the bandwidth for the tree links to support wider links at the root of the tree and forwarding links at the leaf.

III. Physical Design

During the first few decades of high performance processor design (for which domain specific accelerators are a subset) there was an emphasis on custom design to meet PPA targets [37]. However, there has been a considerable shift towards using fully automated synthesis, placement and routing tools to minimize design effort and turnaround time while maintaining high frequencies and area efficiency. This can be seen for even the most performance critical designs that demand high frequencies [38]. Furthermore, many accelerators for edge inference, including those by Nvidia and ARM, are either provided as open-source or licensed soft IP. These IP lend themselves very well to Automated-Place-and-Route (APR) [39].

DNN accelerators have relatively simple dataflow and associated floorplans, compared to CPUs. An evaluation of the floorplan of the first version of Google's TPU, as shown in Table III shows that most of the area is allocated to compute and memory to feed the compute and store the results of the compute [40].

TABLE III: Area Allocation for a Typical Accelerator

Functional Unit	Area Allocation
MAC Core and Activation Pipeline	30%
Unified Buffer and Accumulators	37%
Control	2%
DRAM/PCIe and other misc I/O	10%
Other	21%

As in the case of CPUs, process scaling and design refinement can provide substantial frequency, bandwidth and memory capacity uplift [41]. However, difference in process scaling rates between logic and

memory will present a significant challenge for ML accelerators going forward. In deep-submicron technologies, SRAM bitcells (on-chip memories), are scaling at a much smaller rate than the logic. For example, the transition from 5nm to 3nm nodes represents $\sim$1.7$\times$ increase in logic density while only a $\sim$1.2$\times$ increase in SRAM density [42]. Decrease in storage compared to a relatively high increase in computational resources may limit the performance benefits as the accelerators may become memory bandwidth-limited [40].

Physical design challenges for accelerators stem from their scalability aspect. In order to design an accelerator with a desired performance and power target, the number of PEs, and the amount of on-chip storage and how to interconnect them into an array need to be decided. From an interconnect scalability perspective, we hit a hard limit on the number of PEs that can be interconnected with a certain amount of memory within an array. Any requirement to scale beyond this limit requires hardening the array and tiling it further. The number of tiles and the data distribution network to interconnect them and keep them fully utilized give rise to interesting physical challenges that are unique to accelerators. These hurdles arise due to the large area of the chip that needs to be spanned introducing clock skew problems, wiring congestion and difficulty in converging on timing.

IV. Future: Emerging Trends and Technologies

We conclude by describing some emerging techniques that are yet to be adopted by mainstream DNN accelerators but have the potential to drastically improve their efficiency in the near future:

i) *Approximate computing* - DNN algorithms have been known to be resilient to approximations introduced during computation [43]. As a result, approximate computing techniques have been shown to be highly effective in improving the performance as well as energy efficiency of DNN accelerators. Although a lot of these techniques are yet to be adopted for mainstream commercially available DNN accelerators, there already exists a significant body of research work to show its potential impact on system-level energy consumption [44]. Apart from the usual approximation techniques such as channel pruning, quantization, operator strength reduction, *etc.*, examples of other feasible software/hardware based approximation techniques include novel methods such as lossy compression [45], activation prediction using a helper network [46] or using *apriori* knowledge of already trained weights [47], and even applying synergistic approximations across multiple subsystems [48].

ii) *Compute in DRAM* - Although compute-in-memory has shown great potentials due to high memory bandwidth, finite storage and generalization as described in Section II-A are some key obstacles for being adopted by industry. Recent studies such as [49] explored having DNN acceleration within the DRAM die which helps alleviate the storage issue while seeking ways to support wider range of networks. Although DRAM process poses challenges to logic design, the solution can draw great interest if proven to work in commercial settings.

iii) *Runtime flexibility for efficiency* - Accelerators are often part of a larger compute system. Retaining the ability to efficiently distribute the compute across heterogeneous subsystems and memory hierarchy and catering to dynamic control flow present in modern network architectures is critical. The hardware-software co-design of such systems will dictate the mapping of workloads to the accelerator [50]. It is important for accelerator designers to take into account the efficiency of the entire system in addition to executing the work assigned to the accelerator efficiently.

References

[1] B. Zoph *et al.*, "Learning transferable architectures for scalable image recognition," in *Proc. CVPR*, 2018.
[2] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Comm. of the ACM*, vol. 60, no. 6, 2017.
[3] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016.
[4] A. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *ArXiv*, 2017.
[5] D. Hernandez and T. B. Brown, "Measuring the Algorithmic Efficiency of Neural Networks," *arXiv preprint*, 2020.
[6] A. Reuther *et al.*, "Survey of Machine Learning Accelerators," 2020.
[7] M. Horowitz, "1.1 Computing's energy problem (and what we can do about it)," in *Proc. ISSCC*, 2014.
[8] Y.-H. Chen *et al.*, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *SIGARCH Comput. Archit. News*, vol. 44, no. 3, 2016.
[9] Y. Chen *et al.*, "Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, 2019.
[10] L. Wang *et al.*, "Adversarial robustness of pruned neural networks," in *ICLR Workshop*, 2018.
[11] Y. Chen *et al.*, "A Survey of Accelerator Architectures for Deep Neural Networks," *Engineering*, vol. 6, no. 3, 2020.
[12] "Edge TPU," https://cloud.google.com/edge-tpu/, 2019.
[13] H. Kung, "Let's design algorithms for VLSI systems," 1979.
[14] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Comp. Arch. News*, vol. 44, no. 3, 2016.
[15] V. Sze *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, 2017.
[16] X. Yang *et al.*, "Dnn dataflow choice is overrated," *ArXiv*, 2018.
[17] R. Banakar *et al.*, "Scratchpad Memory: Design Alternative for Cache on-Chip Memory in Embedded Systems," in *Proc. CODES*, 2002.
[18] P. Angshuman *et al.*, "SCNN: An Accelerator for Compressed-Sparse Convolutional Neural Networks," in *Proc. ISCA*, 2017.
[19] A. Parashar *et al.*, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *Proc. ISPASS*, 2019.
[20] H. Kwon *et al.*, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," *Proc. MICRO*, vol. 40, no. 3, 2020.
[21] C. Farabet *et al.*, "NeuFlow: A runtime reconfigurable dataflow processor for vision," in *CVPR Workshops*, 2011.
[22] Z. Du *et al.*, "ShiDianNao: Shifting vision processing closer to the sensor," in *Proc. ISCA*, 2015.
[23] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *Proc. ISCA*, 2018.
[24] Y. He *et al.*, "Channel pruning for accelerating very deep neural networks," in *Proc. ICCV*, 2017, pp. 1398–1406.
[25] S. Han *et al.*, "Learning both weights and connections for efficient neural networks," in *Proc. NIPS-28*, 2015, p. 1135–1143.
[26] X. Xu *et al.*, "Hybrid pruning: Thinner sparse networks for fast inference on edge devices," *CoRR*, vol. abs/1811.00482, 2018.
[27] J. Albericio *et al.*, "Cnvlutin: Ineffectual-Neuron-Free Deep Neural Network Computing," in *Proc. ISCA*, 2016.
[28] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," *ACM SIGARCH Comp. Arch. News*, vol. 44, no. 3, 2016.
[29] Nvidia, "Nvidia a100 tensor core gpu architecture," 2020.
[30] Intel, "Intel keembay," https://newsroom.intel.com/wp-content/uploads/sites/11/2019/11/intel-ai-summit-keynote-slides.pdf.
[31] X. Zhou *et al.*, "Cambricon-S: Addressing Irregularity in Sparse Neural Networks through A Cooperative Software/Hardware Approach," in *Proc. MICRO*, 2018.
[32] Y. Lu *et al.*, "SparseNN: A performance-efficient accelerator for large-scale sparse neural networks," *Intl. Jour. of Parl. Prog.*, vol. 46, no. 4, 2018.
[33] M. Elhoushi *et al.*, "DeepShift: Towards Multiplication-Less Neural Networks," 2020.
[34] T. Moreau *et al.*, "A Hardware–Software Blueprint for Flexible Deep Learning Specialization," *Proc. MICRO*, vol. 39, no. 5, 2019.
[35] K. Shuler, "AI Chips: NoC Interconnect IP Solves Three Design Challenges," Semi. Engg., 2019.
[36] H. Kwon *et al.*, "MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects," *ACM SIGPLAN Notices*, vol. 53, no. 2, 2018.
[37] J. Warnock *et al.*, "Circuit and Physical Design Implementation of the Microprocessor Chip for the zEnterprise System," *IEEE JSSC*, vol. 47, no. 1, 2012.
[38] T. Singh *et al.*, "2.1 Zen 2: The AMD 7nm Energy-Efficient High-Performance x86-64 Microprocessor Core," in *Proc. ISSCC*, 2020.
[39] Nvidia, "The Nvidia Deep Learning Accelerator," http://nvdla.org/primer.html, 2018.
[40] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ISCA*, 2017.
[41] T. Norrie *et al.*, "Google's Training Chips Revealed: TPUv2 and TPUv3," in *Hot Chips*, 2020.
[42] TSMC, "Technology Symposium and Open Innovation Platform Ecosystem Forum," 2020.
[43] C. Chen *et al.*, "Exploiting approximate computing for deep learning acceleration," in *Proc. DATE*, 2018, pp. 821–826.
[44] A. Raha and V. Raghunathan, "Approximating beyond the processor: Exploring full-system energy-accuracy tradeoffs in a smart camera system," *IEEE TVLSI*, pp. 2884–2897, 2018.
[45] A. Ranjan *et al.*, "Approximate Memory Compression," *IEEE TVLSI*, vol. 28, no. 4, 2020.
[46] S. Cao *et al.*, "SeerNet: Predicting Convolutional Neural Network Feature-Map Sparsity Through Low-Bit Quantization," in *Proc. CVPR*, 2019.
[47] V. Akhlaghi *et al.*, "SnaPEA: Predictive Early Activation for Reducing Computation in Deep Convolutional Neural Networks," in *Proc. ISCA*, 2018.
[48] S. K. Ghosh *et al.*, "Approximate inference systems (AxIS) end-to-end approximations for energy-efficient inference at the edge," in *Proc. ISLPED*, 2020.
[49] S. Cho *et al.*, "Mcdram v2: In-dynamic random access memory systolic array accelerator to address the large model problem in deep neural networks on the edge," *IEEE Access*, pp. 135 223–135 243, 2020.
[50] Y. Yu *et al.*, "Dynamic control flow in large-scale machine learning," in *Proc. EuroSys*, 2018.