Systolic Array based Hardware Architectures for Nueral Networks

Odrika Iqbal and Saquib Siddiqui

I. Introduction

As we are approaching the saturation phase of Moore's Law and Dennard Scaling which we had a heavy reliance on for performance and energy efficiency improvement, over the last decade there has been a shift toward the design of custom and domain specific processors and systems on chip. Among the existing architectures, one of the most promising solutions is the systolic array based architecture. Invented by H. T. Kung and Charles Leiserson in 1979, this architecture has seen a widespread adoption over the last few years in most existing applications related to artificial intelligence, computer vision and many other related tasks. Systolic arrays are fixed and identical nodes commonly referred to as Processing Elements (PEs) that perform primitive tasks such as multiply and accumulate also known as MAC operations. This is useful to perform various tasks such as matrix multiplication and convolution which are essentially building block operations for neural networks. These nodes are further connected using programmable interconnects which can help program data access patterns and hence provide various degrees of parallelism.

In this study, we are going to focus on the use of systolic arrays for use in hardware implementation and subsequent acceleration for Neural Networks(NNs). Before we dive into the specifics of the various works in this field, we need to examine the workload under consideration which is the implementation of neural networks or more specifically Deep Neural Networks (DNNs) in its various flavors such as Feed Forward NNs, Convolutional NNs and Graph NNs, among others. It is well known that the building block of any neural network is a perceptron which is essentially a multiply and accumulate unit. This knowledge is a critical motivation to map them onto systolic array based architectures. We also know that NNs are composed of various configurations of layers with each layer consisting of a given number of perceptrons which is predetermined. In this study we will look at implementation approaches of various types of neural networks and also examine the performance parameters of the various implementations with both a qualitative as well as quantitative approach

II. Reconfigurable Network on Chip interconnects for DNN

While there has been a significant development in performance capabilities of PEs in systolic array processors there has not been an equivalent stride in development of interconnects between PEs. This problem becomes more chronic when we start implementing DNNs on these processors. This is because DNNs are known to be sparse in nature which requires hardware to be adaptable to bandwidth and data types. The existing conventional Network on Chip (NoCs) interconnects are too rigid in their design and hence don't provide much flexibility. It turns out that the data movement through these NOCs suffers from latency because of this inherent rigid design which does not adopt to changes in bandwidth. This work [1] presents a Hierarchical Mesh network HM NoCs which provide a flexible architecture to adopt to bandwidth and hence improve performance. This also provide Single Instruction Multiple Datapath (SIMD) support which is highly desirable for better performance. They are able to achieve about 12x and 2x speed and energy improvement respectively on MobileNet dataset.

A. Introduction

The current work [1] is the second version of their previous architecture [2] where they have improved the capabilities in terms of handling both sparse and dense DNN models while improving both the energy efficiency and throughput. The overall top level architecture is as shown in Figure1 which contains a 8X2 PE cluster array and the further each PE cluster contains a 3X4 PE array. In case of DNNs there are three important data types that interplay namely Input Activation's (iacts), weights and partial sums(psums) which have a dependence on the layer shape and size within a DNN model and subsequently from a hardware perspective they have an effect on the data reuse. Data reuse is critical as it reflects the use of MACs on a given part of data. The above factors vary extensively as we switch from dense to sparse models which are the two major classifications of DNN models and design of hardware that is flexible, adapts and recognizes this change in the DNN model type is critical for higher performance and energy efficiency across various models.

B. Architecture

1) Flexible Hierarchical Mesh Network on Chip Interconnects: To achieve an architecture that offers flexibility, the authors propose a hierarchical structure where the PEs and global buffers(GLB) are grouped together into clusters as shown in Figure1. Another important component is the router cluster that helps with varied degrees of routing of data from either off-chip or from the GLBs to the PEs. It is further found that depending on the data spatial reuse(MACs in the PEs operating on same set of data) the off-chip bandwidth is

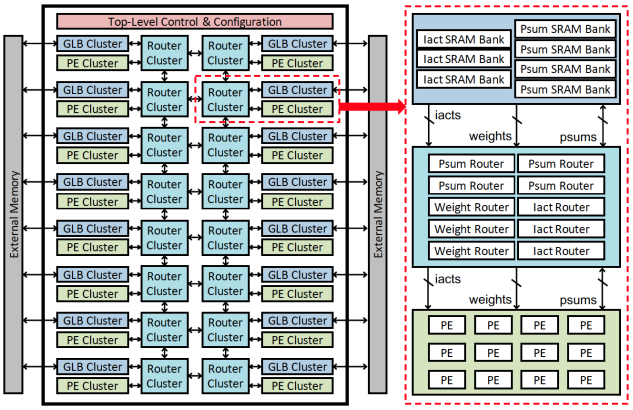

Fig. 1. Eyeriss v2 top-level architecture

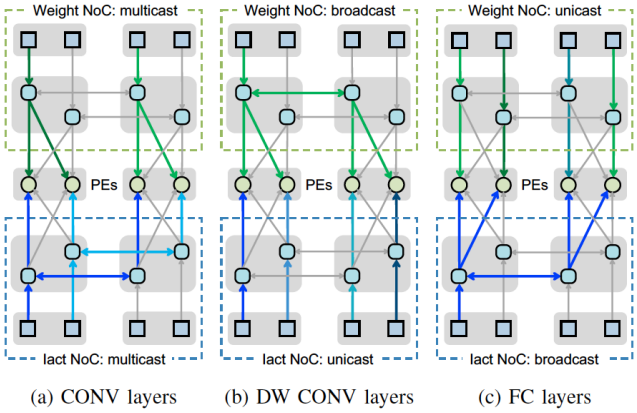

Fig. 3. HM-NoC for various layers : CONV, DP CONV and FC layers

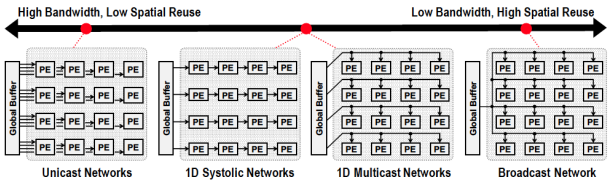

Fig. 2. Relation between data reuse and bandwidth

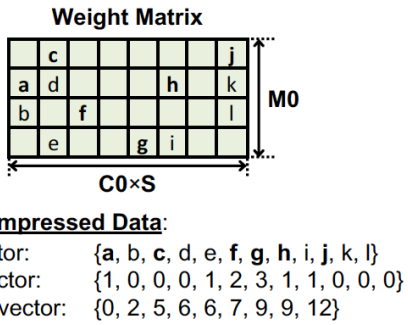

Fig. 4. Compressed Sparse Column compression

accordingly affected. In Figure2 we can observe that when the spatial reuse is low, bandwidth is high and we need a unicast network while at the other end of the spectrum when the spatial reuse is high, bandwidth is low it requires broadcast networks. The critical takeaway from the above is that the interconnects need to aware of the above pattern and accordingly route data to achieve high performance in the various scenarios presented at varying stages of the DNN model. This is where hierarchical NoC are useful as they can adapt the routing for various bandwidth scenarios. In case of a DNN the three major types of layers are Convolutional (CONV) layers, Depth-wise (DP)CONV layers and Fully Connected layers. In each case there are variable data reuse and bandwidth scenarios, for instance in CONV layers data reuse is plenty so both HM-NoC for iacts and weights can configured to multicast mode as shown in Figure3. The NoC are arranged accordingly for DP CONV as well as FC layers. Subsequently the authors develop specialized HM-NoC for the three data types : iacts, psums and weights. As discussed above the most critical component of the design is the router cluster that has specialized capabilities(additional circuitry) to facilitate the three data types in varying scenarios of data reuse and bandwidth conditions. The authors have also done some scalability exploration where the number of PEs within a cluster is fixed and the overall PE cluster size is scaled up. It is observed that the design scales well with higher dimensions of PEs. This is in part due to the hierarchical approach taken where the NoC are designed to be adaptive to bandwidth requirements. However when the number of PEs is very large, the off-chip bandwidth becomes a bottleneck.

2) Sparse Processing with SIMD support: In the previous work there was support for sparsity of iacts by use of clock gating which only leads to energy efficiency improvements. Hence this work both throughput and energy efficiency is optimized for both iacts and weights by using data compression until the PEs. The authors make use of Compressed sparse column (CSC) compression format which contains data, count and address vector as shown in Figure 4 Here the count value indicates the amount of address change between non zero value and the address indicates the data length of each segment. As both iacts and weights are in the CSC format, processing can happen directly in the CSC domain leading to performance and energy efficiency improvements. The Figure5 shows the architecture that contains two sets of address and data(stores both data and count vectors) scratch pads that contain the data in the above CSC format for both iacts and weights. In the overall pipeline it first checks for non-zero iacts and hence subsequently down the pipeline only non-zero weights will be read. The address SPads precede the data SPads to resolve any read dependencies. The above mechanism enables the architecture to handle various DNN models with sparsity and still continue to improve performance and energy efficiency. The PEs also have ability to switch to uncompressed domain whenever there is very low sparsity hence making the design flexible to both types of DNNs. Further there is extension for SIMD support as well by increasing the word

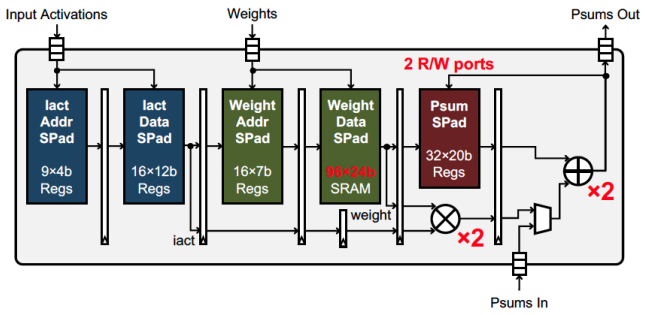

Fig. 5. Architecture to support Sparse Processing

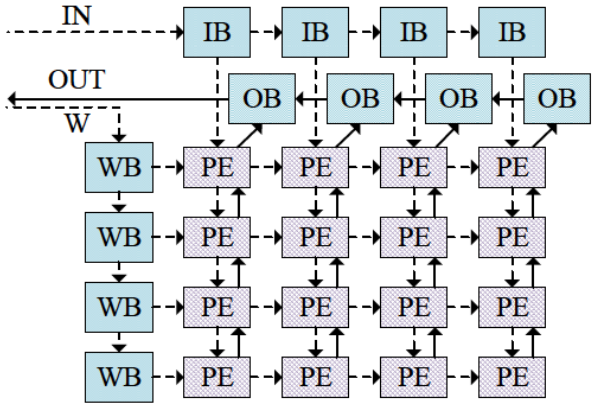

Fig. 6. Systolic 2-D architecture

length of the weight data SPad. This basically implies that in a given cycle two MAC operations are performed using two weights for a given iact.

C. Results and Discussion

The above work is towards extending support of systolic array processors to compact(also known as dense) as well sparse DNN models. This is extremely useful for wide array of applications as the same processor can accommodate both types of models. This has been possible through an intelligent design of NoC to handle various Bandwidth scenarios. Further there has been an extended support for sparse processing using compression coding format which also offers SIMD support. In terms of quantitative results the authors have bettered their own previous work [2] by factor of 42.5x and 11.3x in terms of throughput and energy efficiency for AlexNet and 16.x and 2.5x improvement for MobileNet. The key insights obtained from the above work is the identification of the bottleneck of NoC in the first place and its impact on throughput and energy efficiency. While most recent works in the domain of custom processor design emphasize on improving the computational component of their design(i.e., PE itself) this work has opened up a new avenue of exploration which is the data movement component through analysis and improvements of NoC interconnects. Further rethinking the interconnect to introduce a notion of flexibility implies the utility of the processor has been extended making it much more efficacious. Another insight has been the fundamental division of operations involved within a DNN into iacts, weights and psums. This has given the hardware designer a better control of the operations down to the MAC level and even finer details. This helps in the design phase as there is a higher degree of awareness of the operations which facilitates better optimizations. While the above work has been a useful deep dive into NoCs and there effect on throughput there can also be some trade off studies that discuss the cost of improving NoCs in terms of latency and area constraint. As we know these flexible capabilities come as the cost of added hardware which might not be possible for embedded and IoT applications where the size and weight are important factors besides the performance and energy efficiency. This will be useful in helping future designers make informed choices while being aware of the trade offs at play. Also by using compression they impose an added overhead of compressing the data in the first place which might not be plausible in certain scenarios. Further the class of models targeted : DNNs is only a small subset of the expanding subset of models. It would be useful to see the extension of these processors to implement other types of networks such as Graph Neural Networks, Recurrent Neural Networks and various types of encoders. While that seems like a large class of problems extending capabilities should also extend generalization which will lead to a net positive impact of design complexity.

III. Automated Systolic Array Architecture Synthesis

In regards to mapping of CNNs onto systolic array based architecture there are many mapping decisions to be made in regards to PE mapping, shape selection and data reuse strategies. This work [3] provides an analytical model for performance and resource utilization to help develop an automatic design space exploration framework. In this regard a architecture abstraction is presented that does a loop tiling based representation that helps aid in the high level modeling. Further a resource utilization and performance model is proposed and then all of them are combined to produce a combined analysis model. Hence they are able to perform a design space exploration using the above abstraction and modularity. The space is divided into two regions : architecture based and circuit based. All of the above steps are integrated and an end to end framework is proposed which greatly simplifies the entire mapping hence providing better optimization opportunities. This work serves a great test bed to evaluate exhaustively the feasibility of the design and hence perform subsequent optimizations to improve mappings. The work uses a user friendly high level synthesis based approach that helps with better user understanding and familiarity.

A. Systolic Array Architecture and Challenges

In [3] the authors develop a 2-D array architecture for CNNs which is depicted in Figure 6. A few highlights of this architecture are as follows :

1) Low Routing Complexity : As this 2-D architecture aligns well the inherent 2-D structure of FPGA layout.

2) Single Instruction Multiple Data path (SIMD) support within each PE using the inter-DSP accumulation interconnects.
3) Massive Parallelization within the CNN : As there are local interconnects between the PEs and data transfer happens in a shifting manner which helps in splitting the global interconnect which has a large fan-out into small local interconnects between neighbouring PEs. The shifting data transfer eliminates the multiplexes required.

In terms of execution phase the weights and inputs are loaded into the Input Buffers(IBs) and Weight Buffers (WBs) as shown in Figure 6. Subsequently in following cycles the weights and inputs propagate across the PEs horizontally and vertically respectively. This data access pattern helps with much better data reuse and localization. The PEs also serve as intermediate routers for subsequent PEs in both dimensions. While the above architecture is able to achieve massive parallelism there are certain challenges associated with the mappings of CNNs.

1) Feasible Mapping : There needs to be an appropriate selection of the mappings that ensures proper data availability at specific locations in the PE array in every cycle. As systolic array offers data reuse in both dimensions the corresponding weights and Input arrays need to have high data reuse. While output buffer(OB) needs to carry the result of accumulation.
2) PE array shape : The size of each dimension impacts the performance in terms of DSPs, clock frequency and the DSP efficiency. It is observed that configuration should be compatible with the dimension of the Convolutional layer being mapped.
3) Data reuse strategy : Need to identify proper tiling sizes to achieve extensive data reuse. The problem here is that there are many orders of magnitude design options to choose from in terms of the various on-chip memory utilization and off-chip bandwidth saving.

```
// outer loops
L0: for (Lo = 0; Lo < l0; Lo++)
...
Ln: for (Ln = 0; Ln < ln; Ln++)
// middle loops
S0: for (So = 0; So < s0; So++)
...
Sn: for (Sn = 0; Sn < sn; Sn++)
// inner loops
T0: for (T0 = 0; T0 < t0; T0++)
T1: for (T1 = 0; T1 < t1; T1++)
T2: for (T2 = 0; T2 < t2; T2++)
  // Orignal loop body
```

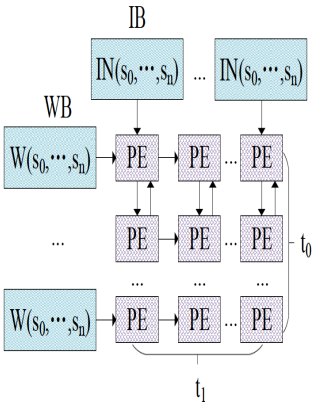

Fig. 7. Loop tiling representation

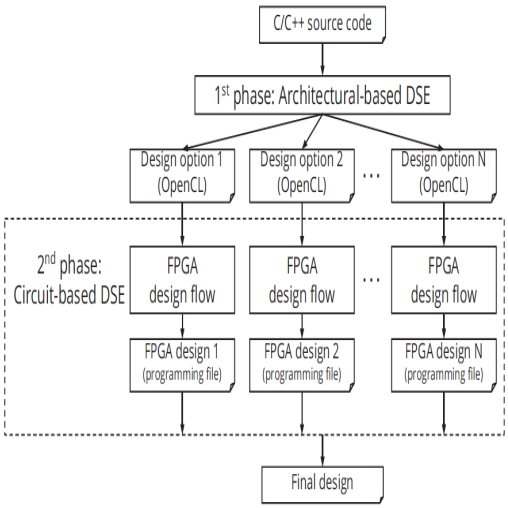

Fig. 8. Design Space Exploration

B. Analytical Models

To solve the above challenges the authors devise the following analytical models that aim to resolve each of the above issues.

1) Architecture Abstraction: The above work proposes a loop tiling representation as shown in Figure 7 which connects architecture to high-level program code. Here the execution is performed block by block sequentially where the outer loops($\mathbf{L_0} - \mathbf{L_n}$) are the blocks. Once data has been loaded onto the IB and WB the middle loops($\mathbf{S_0} - \mathbf{S_n}$) represent the sequential process of feeding data from the buffers into the PE array. The bounds in this loop determine the size of the reuse buffers as the weights and inputs are reused by all the PEs. The final fine-grained parallel execution occurs in the PE array represented by inner loops ($\mathbf{T_0} - \mathbf{T_2}$)

2) Problem Formulation: Considering the remaining factors of feasible mapping(which uses a polyhedral model), resource utilization and performance modeling the overall optimization is formulated as a combination of two sub problems :

1) Given a nested loop condition based on CNN finding the set that contains all feasible systolic array configurations.
2) Given a systolic array configuration finding the optimal bounds of the middle loops to achieve maximum throughput.

3) Design Space Exploration: To minimize the search space of the above formulated problems the authors propose a two phase exploration process as shown in Figure8. In the architectural phase design space is reduced by the considering low DSP utilization which reduces the space by 2.5x. The space is further reduced by leveraging the fact that the BRAM sizes being implemented are an exponent of two. This additionally reduces the design space by roughly a factor of 17.5x.

C. Results and Discussion

This work has used a unified systolic array design for configuration for all convolutional layers within a CNN model. The design is selected based on the above analytical model

and then further classifies them based on estimated throughput(which is under 2% of on-board results). The authors tested their models on AlexNet and VGG16. It observed that the average throughput is roughly 400 and 550 Giga Operations per second (Gops)for AlexNet and VGG16 respectively. Some key takeaways from the above work are :

1) End to End Implementation Simulation Framework : A High-Level Synthesis based end to end framework is proposed that is easy to understand and implement
2) Analytical model with design space exploration technique : The model includes the various factors that need to be accounted for efficient implementation of CNN models. Further an exploration scheme is presented that minimizes the search space hence enabling rapid exploration.

While the above work has addressed a challenging aspect of systolic array implementation of CNNs it still has some areas of improvement that can help further improve the design of systolic array based processors. The above design accounts for the DSP utilization but that is not a direct reflection of the efficient utilization of each PE. This is because the data arriving itself might contain zeros in which case the design should intelligently adapt circuitry to avoid using DSPs in the first place. There is no mention of the interconnects involved in the PEs and what is the latency introduced by the data movement which is an important factor besides the computation itself. Further the work takes a unified design approach which might not always be the best option. For example in case of MobileNet which is highly sparse the above approach will not scale with that and the performance will subsequently decrease. The off chip memory exchanges(loading data into the weights and input buffers) are not reported. These memory movement operations are extremely expensive and need to be reported to get a more realistic picture of the overall result obtained. Also the batch size has not been discussed and its a very important factor in determining overall performance. In conclusion, the above work has helped with the required problem formulation and design exploration for the mapping of CNNs onto systolic arrays. But it has gaps that need to filled in regards to overall mapping approach as well reporting of the corresponding result metrics.

IV. Performance and Energy efficiency evaluation

When we evaluate the performance of DNN mapping on Systolic Array based processors there are various metrics that need to examined to accurately report the corresponding performance in terms of throughput and energy efficiency.In [4] the authors first provide a categorical approach to this problem. While throughput is an indicator of the performance of the processors there are several related metrics that remain unreported in most works and they are critical to get a overall picture of the performance and develop a much better realistic appreciation of the results. While reporting performance there is an implicit assumption that the PEs in the processors will always have the best possible performance. While this a possible scenario it has a dependence on many implicit factors such as data access patterns, interconnect speed, data delivery, data type and parameters related to the neural network itself. These factors are not accounted in significant detail hence leaving results incomplete and non realistic. To combat this, there is need to better analyze the DNN model itself as it has dependence on the performance (which might not be obvious at times). Further its observed that the major culprit of latency is not the computation itself but the data movement which accounts for much higher chunk of the latency introduced. This problem requires a deeper analysis into the data movement and delivery within and among PE arrays. Further the energy efficiency also has a heavy dependence on the data movement, but most existing works don't account for that and have a reliance on only computation related energy cost. For more realistic results the energy cost related to data movement needs to be reported possibly in terms of the power consumption of the DRAM or off chip memory as these are expensive tasks and account for a significant power usage. The factors that affect the performance can be divided into four categories

A. Accuracy

Accuracy is an indicator used for qualitative analysis of results. In light of DNNs this can depend on the task being performed. The definition of accuracy varies with tasks. In image classification tasks accuracy is a percentage of correctly classified images while in object detection tasks accuracy is reported as the mean average precision which is related to the trade offs between true positives, false positives and false negatives. The factors that affect the above metric include difficulty of task and data set itself. For example classification on ImageNet data set is more complex than on MNIST data set which is an instance of the effect of data set on the results. While object detection is more difficult than classification which is case of difficulty of task itself. The reason the above factors are relevant is because the corresponding DNNs models for each tasks will have added degree of complexity(i.e., in terms of MAC operations and distinct weight features) which impacts the efficiency of hardware that processes the model. A platform that can assist in reporting accuracy correctly is *MLPerf* which provides normalized results for various common DNN models and helps with fair comparisons.

B. Throughput and Latency

Throughput which is used to indicate the amount of data that can be processed in a given time period is often a critical metric for any given application. Latency is the time interval between input data's arrival into the system and the generation of the result. With the ever increasing conglomerate of visual data, its desirable to have a higher throughput. As we have been shifting towards real time applications such as augmented and mixed reality low latency is becoming more desirable. It is often assumed that there is direct correlation between throughput and latency and that they are inversely proportional. It turns out that this assumption does not necessarily hold in true for certain scenarios. For instance when we use batching of input data (i.e., combining multiple images or frames together for processing) to increase throughput the latency does necessarily

scale down but actually increases which then conflicts with the reported throughput as the latency dominates. We can mathematically formulate and combine both throughput(which in case of DNNs can also be represented by Inferences per second) and latency which is measured in seconds then we obtain the following formula :

$$\frac{Inferences}{second} = \frac{operations}{second} \times \frac{1}{\frac{operations}{inference}} \quad (1)$$

An interesting observation is that the first term in the equation has a dependence on DNN hardware as well as the model. While the second term only depends on DNN model only. In case of systolic array architecture the first term has a direct dependence on the performance of Processing Elements(PEs) in the design. While the trivial observation would be that performance scales linearly as we increase the number of PEs this observation does not account for several factors. It turns out that there is a tradeoff between the number of PEs in the design and the on-chip data storage. The performance of each PE depends on the availability and constant flow of data (which depends on the on-chip storage) that affects it utilization. Here utilization can be thought of a variable factor and its not necessarily unity(100%) at all times which might be the naive assumption. Also increasing number of PEs might not imply an increase in performance as we have to sacrifice on-chip storage to accommodate more PEs under a given constraint. To study the dynamics of the above tradeoff a procedural method called Eyeexam is devised which is essentially inspired by roofline model(which relates Performance to Operational Intensity). In the Eyeexam the initial condition assumes infinite parallelism, storage and Bandwidth. Then it incrementally imposes the following design constraints :

1) Layer metrics(Size and Shape) which is essentially constraint based on the workload involved.
2) Dataflow
3) Number of PEs
4) Dimensions of the PE arrays which is the spatial arrangement of the PEs
5) Storage Bounds which makes the buffers finite in size
6) Bandwidth of Data
7) Variation of Bandwidth with time for various data access patterns

The two key takeaways from the above analysis is that number of MAC operations in the DNN model and number of PEs is not a direct indicator of the overall performance. The latter is because its not necessary that throughout the operation all the PEs are being utilized at their full capacity. While the former implies that each operation is not equal within a model in terms of computational complexity and the hardware should recognize this complexity without docking the overall performance itself.

C. Energy Efficiency and Power Consumption

The overall energy consumption is chiefly classified into two categories namely consumption due to data movement and computation itself. While the computation consumption is calculated from the MAC operations, data movement consumption is calculated by on-chip and off-chip memory accesses. It is widely observed that computation consumption is reported but the data movement part is left out is most works related to systolic array based designs. However it turns out that data movement accounts for a huge amount of the power consumption and not reporting the above leads to anomalies in the efficiency of the system. For improving the computation consumption we can employ fixed point data type format wherever possible as opposed to operating in the computationally heavy floating point domain. To improve the data movement costs better on-chip caching and reuse techniques can be employed that reduces the external memory access related energy cost. In conclusion its vital to report the energy expenditure of off-chip memory as well besides the consumption of the chip which only contains the computation consumption and partial data movement consumption.

D. Hardware Cost , Flexibility and Scalability

One of the most important elements in terms of cost is the chip area and the process technology or node which also drives the availability of on-chip storage and compute in terms of PEs. Further the off-chip BW also has a dependence on chip area within given constraints where increasing one can decrease the other and there has to some trade off analysis to operate at the optimal point. Flexibility refers to the spectrum of DNN models a design can support. This support is classified at a finer level into two levels where initial level involves functional support and the more involved level where there is an expectation of efficiency as well in terms of both throughput and energy consumption. Here we can take the classical analogy of over fitting from neural networks, when a model that over fits the training data it performs poorly on test data. So in this case if the hardware is extremely tuned to a certain DNN model it will perform poorly with different model that introduces new constraints and nuances. To solve this problem there has to be additional hardware that imparts the necessary flexibility to extend support. But this overhead introduces an energy and area cost. Hence additional hardware is warranted if it leads to a net gain over wide spectrum of workloads. Finally Scalability refers to ability to scale up a given hardware design to achieve higher orders of performance and efficiency by multiplying the amount of resources available(in this case number of PEs and the corresponding on-chip storage capacities). Its assumed that performance should scale up linearly with the number of PEs. This has dependence on the batch size in consideration and keeping it(batch size) constant makes it much harder to scale and its referred to as strong scaling while in case when batch size can be increased its referred to as weak scaling which is comparatively less challenging. For energy efficiency this relationship is not necessarily linear as increase in on-chip area can lead to abrupt improvements.

E. Discussion

It clear from the various factors reported above that the design space for processors is extremely large. Hence an

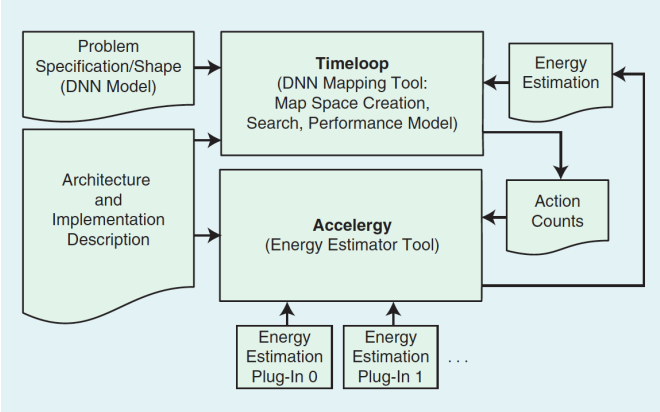

Fig. 9. Energy Estimation Tool

estimation is required based on various factors such as DNN characteristics(model shape, precision, sparsity), throughput and energy efficiency required. A tool that helps with better exploration of these factors is the combination of Timeloop [5] and Accelergy [6] as shown in Figure 9. The tools take in the model specification to generate optimal mapping using analytical performance models and subsequently generate architecture level energy estimates. The above work has clearly factored out the various metrics that interplay in determining the overall performance. It has shown that factoring all of the above parts leads to more realistic and appreciable architecture that can be extended further. With the current direction of DNN models and subsequent design of DNN processors there needs to be unified performance evaluation approach that will help easy adoption of these DNN architectures. It is also very clear that DNN models need to developed in direct integration with the DNN processors themselves in a concurrent fashion rather than sequentially where one is isolated from the other. This is because contrary to assumption DNN model has an impact on the performance of DNN processor which is not explicit at the initial design phase of processors. Also its clear that DNN processor designs that accommodate some notion of flexibility are useful as they have wider utility because of the ever expanding space of DNN models.

V. SCALE-Sim

In [7], the authors propose SCALE-SIM (Systolic CNN Accelerator Simulator) - a simulation framework for systolic array based neural network accelerators. The key benefit of systolic-based architectures is that they facilitate efficiency in dense matrix multiplications and local shifting data movement. Neural networks are notoriously compute heavy and hence, this type of architecture is ideal for neural network based accelerators. It should also be noted that the data shifting and operations inherent in systolic-style systems resonate very closely with the sliding window mult-add operations associated with convolutional neural networks.

The authors note a dearth of tools as means of analyzing key design trade-offs and efficient mapping frameworks for systolic-array based deep neural network (DNN) accelerators. Their proposed simulator SCALE-SIM enables users to access several micro-architectural features along with system integration parameters. This allows for exhaustive design space exploration and makes design optimization feasible on the part of the user. To the author's knowledge, this is the first systolic array simulator of its kind and this shows great promise as a tool that can be useful for both computer architects and machine learning scientists.

A. Simulation Methodology

1) Computation Simulation: SCALE-SIM formulates the DNN accelerator's computational unit as a systolic array. The systolic array is constructed with several Multiply-and-Accumulate (MAC) or units or Processing Elements (PEs) which interconnect in a 2-D mesh. Data flows in through the edges and gets transmitted to the elements in the same row and columns with the help of unidirectional links. The incoming data is kept in an internal register in the current cycle and the MAC unit disseminates it to the outgoing link in the subsequent cycle. This helps optimize memory bandwidth and also enables the exploitation of data reuse for efficient DNN operations.

2) Dataflow Simulation: The dataflow has been defined and characterized the same way as in [1]. Output Stationary (OS), WeightStationary (WS), and Input Stationary (IS) are the dataflows modeled in the SCALE-SIM simulation framework. In this paper, they have been explained with a systolic array based implementation in mind.

Output Stationary (OS) - As the name signifies, each output pixel is assigned a specific processing element (PE), and all operations needed to obtain a particular output pixel are executed on the corresponding PE. All associated operands come streaming in per cycle and the reduction operations are always stationary, with no additional communication requirements between the PEs for generating the output for a given input. For real-life scenarios where resource constraint is a major bottleneck, the output generation process is multiplexed in time such that each generated output is transmitted to memory to free up its PE for computing a different output pixel. Figure 10(a) illustrates the dataflow in motion in SCALE-SIM. The input pixels are streamed in from the left and the filter/weight pixels are streamed in from the top. The PEs belonging to a column are responsible for computing the adjacent pixels in a given channel and each column contains output pixels corresponding to different output channels. ***Weight Stationary (WS)*** - In this type of dataflow, the weights are first pre-filled and remain stationary in the PEs and then the input is streamed in from the left edge every cycle as shown in Figure 10(b). The input here are unrolled convolution windows in time which are operated on by the filter weights for producing the corresponding convolved output. As mentioned before, each weight is mapped column-wise to a unique MAC unit for processing the incoming input data and they remain there until all computations related to them are executed. The partial sums resulting from the operation of the weights on the input data are stored inside the array and are communicated across the PEs in the same column for the purpose of reduction. Note that the reduction requires n number of cycles where n denotes

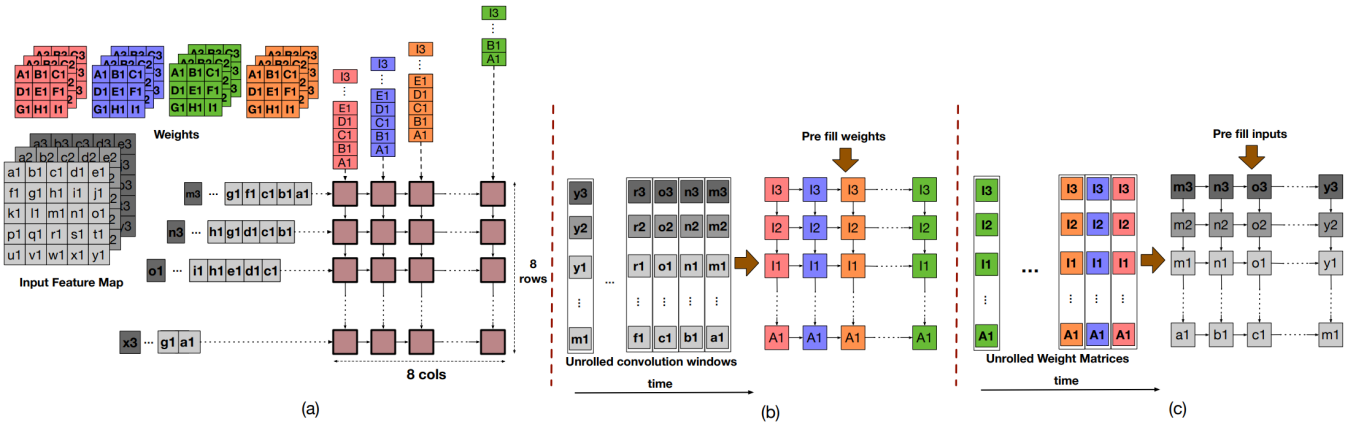

Fig. 10. Illustration depicting the mapping strategies of the three dataflows under investigation - (a) Output Stationary (Os), (b) Weight Stationary (WS), and (c) Input Stationary [7].

the number of partial sums derived for a specific input pixel. After the current set of weights serve their purpose, they are replaced with a new set of weights for the subsequent output generation. Owing to the sequential flow of data wherein the weights are first being transmitted to the PEs and the input pixels are then being streamed in - the SRAM memory overhead is lessened compared to the OS variation of dataflow. The drawback, however, is that the partial sums cannot be erased from memory until the reduction step, which shoots up the implementation cost. ***Input Stationary (IS)*** - This form of dataflow is very similar to the WS variation - the only difference being that here the pixels from the input feature map (IFMAP) remain stationary in the MAC units while the filter pixels are streamed in. Figure 10(c) demonstrates the mapping strategy in an IS dataflow based framework. Each convolution window is allocated a single PE, where a convolution window refers to the group of pixels within an IFMAP that are resposible for generating a single output pixel of the Output Feature Map (OFMAP). In the IS style dataflow, input pixels of a convolution window are streamed in from the top to the PEs arranged column-wise. The filter pixels are then transmitted from the left edge before the system moves on to the reduction step. Reduction takes place column-wise and the convolution windows remain in place until all computations related to them are executed. Then the next batch of convolution window pixels are sent in to replace their previous counterparts to generate the new OFMAP pixel. The nature of IS dataflow being very similar to that of Ws endows it with much of the same benefits as the WS dataflow, i.e. reduced SRAM overhead compared to OS. However, the IS dataflow is shown to have different cost and runtime overheads compared to WS depending on the workload.

3) Memory Simulation: CNN operations are notoriously compute-heavy and require a lot of memory. SCALE-SIM enables joint optimization of the computational unit of the DNN kernels and the memory hierarchy by modeling a parameterizable memory hierarchy. The convolution operations entail data reuse, which is leveraged to optimize the on-chip memory hierarchy design. In particular, the following reuse patterns emerge as a result of the dot product operations that take place between the sliding filter kernel and the IFMAP - 1) each output channel corresponds to a single filter kernel which is used repeatedly to filter the incoming input data, 2) Given a stride smaller than the window size, input data in neighboring windows will have overlapping pixels, and 3) For different output channels, different kernels will be implemented but on the same IFMAP. Although this trend of data reuse enables memory optimization, it also creates complexity in the memory simulation aspect of things, especially considering the fact that data reuse is impacted by the user-defined neural network hyperparameters as well as the selected dataflow. In addition to this, the scratchpad memory also needs to be optimized with the user-defined specifications in mind. As such, the authors make provisions for identifying the optimal memory size as a function of whatever workload the user opts for at a given time. SCALE-SIM partitions the memory into three logical blocks for the IFMAP, filter weights and computed OFMAP. The memory allotted to each partitioned, as stated before, is to be decided by the user. SCALE-SIM models the accelerator memories as double buffers to conceal the SRAM access latency. The double buffered memories come in two sets - the first is the working set which enables the streaming in of data to the array and the other is the idle set which is occupied by data transmitted from the off-chip memory.

4) System Interface Simulation: Isolating the performance optimization on the accelerator side from the total system performance is bound to result in sub-optimal system performance. The authors model the system interface to ensure that the eventual system integration turns out to be seamless and practicable. A significant drawback associated with typical systolic array based compute systems has been addressed here. Real life systems are resource constrained and this should be kept in mind while designing the accelerator. Disregarding the real-life system integration for a later stage may wind up costing the future deployment of the system. For instance, the number of PEs deemed optimal for running the accelerator may not be available in the same number in a real system. Scalability in terms of compute units is a major limitation

of systolic array architectures and the real system interface modeling helps alleviate this problem to a certain extent.

In SCALE-SIM, the DNN accelerator is connected to the system processor in a master-slave configuration as shown in Figure 11. The master (or processor) dictates the slave (or accelerator) by writing instructions to memory mapped registers. After the accelerator is triggered into action by the master, the latter switches gears to execute other tasks. After the completion of the task by the accelerator, it maps the generated output to the memory and alerts the master. Such kind of a communication interface makes it logical to analyze just the system bus interface as a representation of interface with the overall system. There is provision for determining the actual read and write bandwidths of a system bus interface in SCALE-SIM. These metrics can then be passed to a DRAM simulator for creating a realistic and practicable memory model for the entire system. Note that the on-chip memories for a DNN accelerator are scratchpad memories which are not bottle-necked by coherence issues in hardware like caches. Therefore, hardware coherence of memories is not managed using SCALE-SIM.

5) Implementation: The authors make two strong assumptions in their implementation of SCALE-SIM - 1) all PEs undergo active operations at all times without fail and 2) there is no stall in the MAC unit operations, i.e. none of the PEs need to await new data. The reasoning behind these assumptions is that SCALE-SIM is not going to be tested for any particular implementation choice. The SCALE-SIM implementation is executed in the following steps:

1) The data streaming in to the systolic array from the top and left edges are assigned cycle accurate read addresses. SCALE-SIM assures that the systolic array never stalls waiting for new data to come in under the assumption that the system will behave the same way irrespective of the implementation choice. The allotted addresses are essentially the SRAM read traffic for the convolution window and weight matrices depending on the selected dataflow. SCALE-SIM also produces the output trace corresponding to the output matrix, which is basically the SRAM write traffic, seeing as the reduction step takes place within predictable number of clock cycles.

2) Analyzing the traffic traces generated by SCALE-SIM provides us with the latency as well as resource utilization numbers. Latency can be measured by studying how long it takes for the new output trace entry to come in. The SRAM traffic information provides a measure of how much of the resources are to be used up by the DNN accelerator for a given workload.

3) Once the SRAM traffic traces as well as configuration are obtained, SCALE-SIM outputs the DRAM traffic trace for both input and output information transfer.

4) Lastly, the DRAM traces are parsed to estimate the memory bandwidth requirement as well as memory power consumption.

6) User Interface: SCALE-SIM takes in two files as input - a config file with the specifications for the dataflow selection, systolic array structure, memory size, etc. and the DNN topology file with the user-defined specifications for the DNN architecture and related hyperparameters, e.g. the number of layers, channels, strides, etc. The output obtained at the end of simulation also comes is two different files - the first containing the cycle accurate SRAM and DRAM traces in the form of a csv file and the second containing the metric information derived from the memory traces like the latency, memory bandwidth requirement, total data transfer, resource utilization, etc.

B. Design Insights

In this section, we will analyse SCALE-SIM's design reports based on the effect of dataflow, memory sizing, shape of the array, scaling out vs. scaling up, etc.

1) Effect of Dataflow: The authors have made an attempt to study the effect of dataflow on the runtime and energy consumption of a DNN accelerator for seven different workloadS including FasterRCNN, Resnet50, etc. Note that each of the workloads is a different manifestation of a deep neural network. The authors analyze the effect of dataflow on system performance from different perspectives. Firstly, they observe that for different array sizes, the OS dataflow seems to outperform the other two in almost every respect. However, the authors point out that SCALE-SIM does not account for real life stalls that may arise due to the deployment of an OS dataflow based systolic array architecture for specific applications. Further, the OS dataflow may incur higher expense than the other two in terms of area footprint, as the SRAM utilization is significantly higher for the OS type dataflow and it is common knowledge that SRAM banks are expensive in terms of area footprint. With these considerations in mind, if a user wants to choose between the IS and WS dataflows, the size of the array is shown to impact the runtime and execution for different workloads differently for these two dataflows. For a few workloads, the size of the array proves to be of no significance to the system performance and only the chosen hyperparameters affect the latency and resource utilization. Basically, the choice of workload dictates what kind of resources are to be utilized and the dataflow selection should be made based on this information. For instance, for the workloads where the stationary matrix that needs to be mapped are comprised of only a few pixels, the WS or IS dataflows are the optimal choice as the cycles saved in the mapping stage can be spent on actual computation. This is in addition to the fact that WS and IS dataflows entail lower SRAM use. In direct contrast, the OS dataflow has a clear advantage in cases where the stationary matrix mapping will be time consuming. In cases where the weights outnumber the number of output pixels, IS will be a better choice than WS seeing as higher number of weight pixels imply longer mapping time for the stationary filter matrices. WS is a better choice than IS when the weights are outnumbered by the output pixels. One last design insight in terms of the dataflow selection is that the empirical studies conducted over the different workloads reveal that there is no dataflow which will work optimally for all possible combinations of hyperparameters, array sizes and workloads. However, the authors observe that the performance does not diminish by a significant degree by making a sub-optimal dataflow selection. As such, for instances when the

system design does not need to be highly optimized, any of the proposed dataflows will serve reasonably well in implementing an efficient systolic array architecture.

2) Effect of Memory Size: CNN operations offer opportunities for data reuse that can significantly reduce off-chip memory accesses, reducing energy consumption and overall system overhead. In order to exploit the benefits of data reuse, we need to provide sufficient on-board memory. However, design decisions related to memory sizing needs to take into account energy and area footprint trade-offs. The larger the on-chip memory size, the higher the resource utilization as well as energy consumption. SCALE-SIM models the memory based on the user defined specifications and with the end goal of achieving a low latency and low power implementation by zeroing in on the optimal trade-off between data reuse and system overhead.

3) Effect of Shape of the Array: The authors noticed that different dataflows prefer different array shapes for different workloads. In certain cases, WS and OS favor short-wide configurations while for other workloads the preferred array shape changes. If just one array shape is to be chosen, the square aspect ratio is shown to perform reasonably well for all cases. The intuition here is that the array shape has an impact on the mapping time and SRAM utilization. As a result, the shape of the array interacts with the hyperparameters of a workload in different ways for different dataflows.

4) Effect of Scaling Out vs. Scaling Up: Scaling enables higher parallelism and faster computation, and the authors chose to study the effect of scaling up (adding PEs to the systolic array) and scaling out (having more arrays and dividing the computation burden among them). A key observation made by the authors is that the effect of scaling depends on the workload and not on the dataflow selection. This makes intuitive sense, as scaling of the arrays will provide scope for greater parallelism for different workloads in different ways. For six out of the seven workloads, scaling up outperformed scaling out in terms of runtime.

C. Discussion

The authors themselves pointed out that SCALE-SIM operates under the assumption that the systolic array will not be stalled for even a single moment. As a consequence, the real-life runtime execution of a DNN-based process may be longer than anticipated based on the timing predictions obtained via SCALE-SIM. Even though the authors justify this limitation by reminding us that the purpose of SCALE-SIM is to reveal the benefits and drawbacks of the three dataflows without considering any dependencies arising from a specific implementation, it would have been nice to have the added option of investigating how each of these dataflows would delay the corresponing DNN process given any specific implementation choice. This would be a particularly useful add-on for time-critical applications like autonomous driving, augmented reality, etc. where over-promising on the timing performance at the design stage may seriously hamper the eventual system integration and deployment of the accelerated network. Moreover, systolic-array based architectures

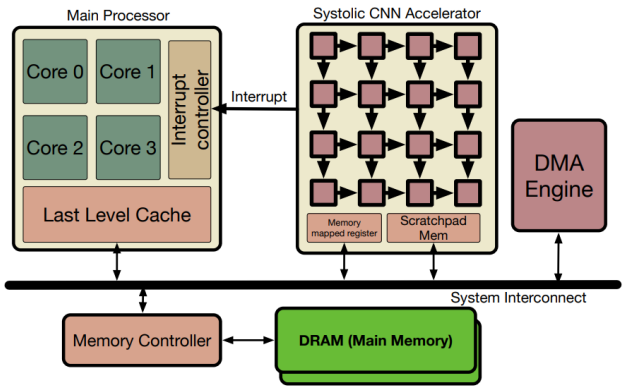

Fig. 11. System integration model for the systolic array based DNN accelerator [7].

are usually highly specialized and demand custom hardware for specific applications. As such, the design choices and associated latencies and memory utilization ought to be a function of the end application and cannot be generalized given the inherent ungeneralizable nature of current systems featuring systolic arrays.

The performance trends rising from different dataflows and user specifications were studied to learn the optimal choices empirically. However, some measure of performance prediction could be incorporated in the SCALE-SIM simulation framework to point the user to the right direction while defining the config and topology files. For instance, if the prediction mechanism deems a certain dataflow to be optimal for a specific workload, SCALE-SIM could limit the user's choice to that dataflow only. This would be of special use to new users not very familiar with the theoretical aspects of DNN accelerators and systolic array dataflows.

Observing Figure 11, we notice that although the accelerator makes use of scratchpad memory in lieu of cache, the main processor (or master) employs cache memory. This raises the question of whether SCALE-SIM takes into account what happens in the event of cache misses at the processor end. The authors argue that simply modeling the system bus interface in place of the overall main system should suffice in the case of master-slave configurations. But it would be more rigorous to generate results with the possibility of potential cache misses on the processor side, as cache misses could very well transpire in a real system.

VI. Optical Hardware for Deep Learning

In [8], the authors propose a freely scalable and reconfigurable optical hardware for neural networks. Electronic processing units need to keep up with the advances taking place in the deep learning domain. The neural network models are becoming increasingly complex with the whole focus being directed towards task accuracy. Although GPUs and DPUs have been popular hardware choices for neural network based operations in the recent past, we face several impediments in scaling networks on these devices including communication expense, thermal concerns, limited power budget and clocking.

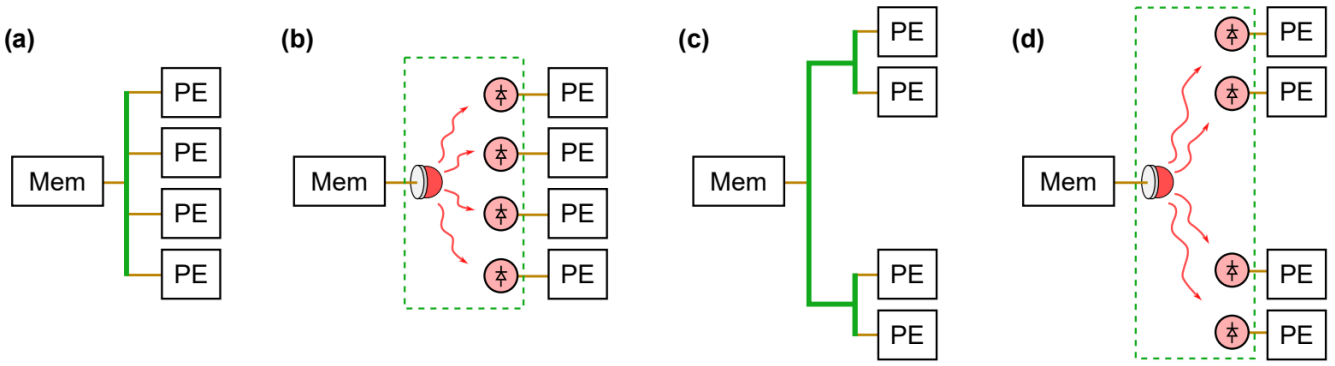

Fig. 12. Illustration depicting how a bit is fanned out from memory to PEs - (a) Transmission via electrical wires, (b) Transmission via optical interconnects (represented by the green lines), (c) Electrical wire transmission to PEs split up into chiplets and (d) Optical interconnect transmission to PEs split up into chiplets [8].

With this in mind, the authors propose a digital-optical neural network (DONN) which features intralayer optical interconnects and reconfigurable input values. Note that energy consumption in optical hardware is approximately independent of the length of the communication channel, which we will call the path independence property in the proposed neural network based framework. As a result of this property, DONN ensures data transmission between a transmitter and arbitrarily arranged receivers without incurring additional expense in terms of energy, thus facilitating greater flexibility in the architecture and allowing for network scaling without violating the inherent power constraints of the device.

In the proposed framework, the DNN kernel encodes data into programmable binary pulses during communication and copying (also called fan-out) to large-scale electronic multiplier arrays. The authors leverage the fact that the optical energy consumption is almost independent of the distance in communication in order to establish communication channels between a single transmitter fanning out to several arbitrarily arranged receivers with both optimized speed and energy efficiency. In the proposed approach, the computational units remain entirely digital with only the interconnects being modeled by optical paths for dataflow. Thus, DONN-based architecture does not necessarily have to trade task performance for gains in energy efficiency.

A. Network Architecture Details

Neural networks comprise of layers of weights or neurons interconnected with one another. The authors propose modeling the interconnecting paths with optical hardware. In this work, the network is implemented only for inference using pretrained weights. This means that the set of network weights are known beforehand and this establishes hard constraints in terms of latency and throughput. In most modern neural network implementations, the majority of the computational burden lies on the fully connected and convolution layers, which entail compute-intensive matrix-matrix multiplication operations and vector-vector inner products. Therefore, parallelizing these operations by deploying a systolic array based architecture assists in overall system optimization. As was already discussed in detail during the review of [7], convolution operations offer opportunities of data reuse. Exploiting these data reuse trends is an efficient way of minimizing execution overhead. Another important factor to keep in mind is that the less the accelerator needs to read data from the main memory (DRAM), the lower will be the energy consumption. Fanning out involves reduced DRAM access and is therefore an optimal way of implementing a DNN accelerator. In this work, the weights are fanned out via interconnects modeled by optical hardware as opposed to electrical transmitters. Weights are activated by sending out on-off optical pulses via the optical connecting paths. This helps alleviate restrictions posed by electrical transmitters such as the need to reduce inter-multiplier spacing and and co-locating multipliers with memory. Instead, optical interconnects automatically reduce the communication cost by virtue of its path independence property.

In the proposed architecture, input data is encoded by optical on-off pulses while weight matrix light sources emit corresponding weight bits which are received by the PEs of the systolic array. The photons generated from the input activations and weight bits are processed in the detector to produce photoelectrons having voltages that are then fed to the electronic compute units. Since both the input data and weights are streamed in, the dataflow under consideration is the OS dataflow (discussed in detail in Section V). The authors point out an energy-latency optimization tactic on top of this where a system bus can be used to implement parallel multipliers for parallel streams of data coming in. Depending on the target task, trade-off analysis of energy/latency optimization vs. resource utilization/area footprint reduction can indicate the optimal choice for the given system constraints.

B. Experiments and Results

To validate the use of optical interconnects in lieu of electric transmitters, experiments were conducted to reveal the bit error rate with a systolic array based DONN network. The transpiring bit error rate was not found to be as low as anticipated, and this was attributed to crosstalk interruptions amongst neighboring optical interconnects arising due to misalignment and blurring effects associated with the camera Bayer filter.

Since the crosstalk is deterministic, it was possible to mitigate the effects by incorporating a crosstalk suppression scheme. Every pixel interacts with its neighbors, and subtracting a fraction of every neighboring pixel from the pixel under consideration is shown to greatly reduce the bit error rate. As a matter of fact, the bit error rate for the red channel turned out to be zero after implementing the crosstalk correction mechanism. The bit error rate also dropped down to negligible numbers for the other channels. The authors then went onto demonstrate results obtained from a proof-of-concept experiment. The experiment entailed digit classification using the MNIST dataset with a three layer, fully-connected DONN network. Without crosstalk correction, the accuracy dropped down by 0.6% compared to ground truth labels. Reducing the number of layers to two saw an accuracy drop of 0.4%. This reinforces the authors' claim that keeping all the compute units in the electronic domain in a DONN network ensures that the task accuracy does not degrade by too much compared to the purely electronic system implementation. Although the interconnects have been constructed with optical elements, the PEs still remain purely electronic and this helps preserve the system performance in terms of task precision. Note that even without crosstalk correction, the MNIST classification task is accomplished with an impressive level of precision by the proposed DONN network. Introducing optical paths for the data to traverse through has reduced the energy consumption by limiting DRAM access via optical fanning out of weights and input activations.

C. Design Insights

Theoretical energy consumption analysis reveals that optical dataflow is advantageous in place of electrical data transfer when the spacing of the computational units is on the order of 10 μm or greater. Compare Figures 12(a) and (c), where the former is depicting a scenario where a bit is transmitted by an electrical wire from memory to the relevant PEs arranged adjacent to one another, and the latter demonstrates what happens if the bit is mapped from memory to PEs forming different blocks or chiplets. Studies reveal that the interconnect energy consumption amplifies dramatically as we move from inter-PE communication among adjacent PEs to inter-chiplet communication where the PEs are arranged in chiplets. In direct contrast, optical interconnects energy consumption does not scale if transitioning from inter-multiplier communication to inter-chiplet communication based systolic array architecture. The optical equivalents of Figures 12(a) and (c) are Figures 12(b) and (d) respectively. The energy consumption associated with the array structure shown in Figure **??**(b) turns out to be equivalent to the one shown in Figure 12(d). This is a significant finding because chiplet-based systems are promising solutions for enhancing the throughput of larger neural network implementations.

In addition to energy optimization, optical interconnects introduce flexibility in deciding placement of memory. Because of the path independence property, the memory can be split up into pieces and distributed across the system as needed irrespective of where the compute units are located. These memories can also be accessed repeatedly for computing in the spatial domain. This spatial separation of computation and memory will enable designers to have the option of choosing smaller sized low-power memory having high bandwidth like DRAM, which undergoes a fabrication process different to ordinary CMOS memory. On top of this, the path independent low-power fan out capability demonstrated by optical interconnects can be leveraged to reduce the system overhead by diminishing the need for memory hierarchy.

An additional benefit promised by DONN networks is for client-server based frameworks where the same inference model can be concurrently used by multiple clients. Optical paths dramatically reduce the overhead that comes with transmitting network weights. As such, even in a multi-client paradigm, the DONN system overhead will be noticeably lower. DONN implementation is also a potential solution for introducing irregular connectivity, such as in graph neural networks which are difficult to map on hardware because of their computational intractability. Graph neural networks come with arbitrary connections and may involve long transmission paths between PEs, with each path standing in for an edge in the graph. Swapping out the electrical interconnects with their optical counterparts may result in a more optimal implementation of such complex networks. For high bit precision tasks like training, the MAC units are larger in size and may be separated by longer distances. DONN, therefore, would be an ideal candidate for training as the associated power consumption would be much lower than what we would expect to get by training an all-electronic systolic array based network. DONN networks could also offer the added benefit of thermal management by distributing the compute units such that they remain far away from each other. Furthermore, this would incur no additional expenses.

D. Discussion

The authors have proposed a way to minimize power consumption while relaxing locality constraints for on-chip memory as well as placement of MAC units. However, a few important issues need to be addressed first. Evaluation of the DONN architecture focused specifically on energy consumption, overlooking the related area overhead and latency. In [9], the authors mention that the latency incurred by introducing optical elements in a system can be substantial - especially since the electronic to optical and optical to electronic conversion of values is not a trivial task and might take up quite a few clock cycles. For added clarification, think of the additional steps that are introduced by employing optical interconnects in place of electrical wires - light needs to be detected and converted to photons, the signal needs to be amplified, the output optical pulse needs to be converted to photoelectrons representing voltages, etc. In addition to this, the authors of [9] also point out that optical hardware is not likely to be manufactured having a size smaller than the wavelength of light. This design constraint does not apply for electronics and the electrical interconnects can be of a size much smaller than that of the wavelength of light. It would be interesting to see how the two network architectures (DNN and DONN) vary in

the context of of area footprint. Not evaluating the DONN in terms of latency and area footprint/resource utilization brings up a lot of questions that need to be answered. Further, the systolic array architecture proposed in this work has not been fabricated. It remains to be seen whether it is actually feasible to map the proposed design onto a chip.

Evaluating the DONN network revealed that the energy consumption crosses the crossover point and goes lower than that obtained with an all-electronic network for path lengths greater than or equal to 5μm. The authors cited the MAC implementation shown in [10] and then used the scaling factor to estimate the size of a MAC manufactured with a 7nm Intel processor in order to get the threshold after which DONN starts massively outperforming typical all-electronic neural networks. This process seems overly simplified and not rigorous. Firstly in [10], the authors used a 28nm commercial standard cell for a 500 MHz target process. Intel introduced their Finfet technology in 22nm dimensions and Superfin Technology in 10nm. Also, commercial MAC units would typically be in 3-5 GHz clock speed. These factors could result in a MAC unit which is substantially smaller than what was approximated in this paper for 7nm. Therefore, the crossover point of 5μm may not be accurate.

VII. Conclusion

In conclusion, the above study looks at the various existing systolic array based architectures and examines the feasibility of mapping NNs on them. The above works have also looked at the common nuances of mapping NNs on systolic array processors and highlighted in detail the role the various factors interplay. [1] provides a solution to the problem of data movement within a systolic array based processor by implementing a flexible NoC structure that helps the architecture deal with wide array of bandwidth requirements to support different layers within DNN models. [3] provides a end-to-end high level synthesis based tool flow that can be extremely useful in case of both design space exploration as well better prototyping before implementing on actual hardware. Also [4] has provided a path towards a more realistic analysis of various performance and energy metrics which is vital to obtaining a better and more realistic understanding of the overall method. [7] provides a useful simulation framework for the design space exploration and trade-offs study of clock accurate DNN accelerators. In [8], an energy efficient solution has been proposed for enabling the scaling of neural networks. Digital-optical hybrid systems show great promise in opening up the avenue for exploring power and memory efficient solutions for the ever increasing complexity of implementing modern neural networks. Note that this type of DNN accelerator facilitates both scalability and reconfigurability in compute heavy neural networks capable of achieving superior task accuracy.

References

[1] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," 2019.

[2] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, pp. 262–263.

[3] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput cnn inference on fpgas," in *Proceedings of the 54th Annual Design Automation Conference 2017*. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3061639.3062207

[4] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "How to evaluate deep neural networks processors tops/w (alone) considered harmful," *IEEE Solid-State Circuits Magazine*, pp. 28–41, 2020.

[5] A. Parashar, P. Raina, Y. S. Shao, Y. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2019, pp. 304–315.

[6] Y. N. Wu, J. S. Emer, and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[7] A. Samajdar, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "Scale-sim: Systolic cnn accelerator simulator," *arXiv preprint arXiv:1811.02883*, 2018.

[8] L. Bernstein, A. Sludds, R. Hamerly, V. Sze, J. Emer, and D. Englund, "Freely scalable and reconfigurable optical hardware for deep learning," *arXiv preprint arXiv:2006.13926*, 2020.

[9] D. A. Miller, "Rationale and challenges for optical interconnects to electronic chips," *Proceedings of the IEEE*, vol. 88, no. 6, pp. 728–749, 2000.

[10] J. Johnson, "Rethinking floating point for deep learning," *arXiv preprint arXiv:1811.01721*, 2018.