Reconfigurable architecture design for ReRAM based DNN Accelerator

Zequan Zhou and Gokhan Memik
Department of Electical & Computer Engineering, Northwestern University, Evanston, IL
Zequanzhou2019@u.northwestern.edu memik@eecs.northwestern.edu

***Abstract*— DNN applications normally have both complex computing patterns and intensive resource utilization features. For computing patterns on data flow, depending on the architecture design of DNN, the message passing within network can be highly irregular and diverse. As for patterns on control flow, the mismatch of granularity can also cause irregular mappings on computing resources. To cope with intensive resource utilization problem, ReRAM array was introduced as the computing unit years ago. While, the regular structure and slow write operations of ReRAM array making it inflexible in run-time. In this paper, we proposed a solution to the above problems with a combination of highly parallel and re-configurable NoC design and re-configurable ReRAM compute unit. At the same time, a series of optimizations were taken on top of design to achieve state of art performance. On the developed architecture simulator, the Re-configurable ReRAM based accelerator architecture can achieve 3.39X in FP16 speedup compared with GPU platform Nvidia P100.**

I. INTRODUCTION

Deep learning neural network (DNN) is becoming more and more popular in diverse application regions due to its generality in design implementation and optimization. However, DNN workloads normally quite computing and communication heavy distinguishing them from traditional workloads. One effective way to solve this problem in the past is to scale the transistors to make more on-chip resources available. However, as the decline of Moore's Law, this method is less attractive and productive. It tends out that more architecture design efforts and novel computing devices exploration needed to solve this problem. ReRAM computing array was then put forward to solve the lack of computing power. To solve the communication bottleneck, more data/communication centric architectures were proposed [1]. While, ReRAM computing-based accelerator suffer from reconfigurability due to in-balance of read/write speed of ReRAM array and in-flexible structure of ReRAM array [2]. On the other hand, data/communication centric architecture may lack enough computing units [1].

The contribution of this work aims to solve the 3 fundamental limits in DNN accelerators namely compute bound, communication bound and reconfigurability. The compute bound is due to multi-dimensional data structure and DNN workload normally heavy in matrix operations and non-linear functions which can't fit well to the on-chip computing resources in traditional architecture [3]. To solve these problems, ReRAM array was implemented as high computing density units to do fast analogue matrix multiplication [2]. As for the communication bound, a light weight customized NoC architecture was taken to supply scalable high-bandwidth, low-latency and low-power communication support which is significant for DNN acceleration because large amount of computing units can become useless when stall for the data. The 3rd limit is the reconfigurability of accelerator, due to the fast evolution pace of DNN, the present algorithms may no longer popular in the future [4]. It is also an intrinsic challenge in computer architecture research that how to achieve both generality and customization rather than just trade-off. In this work, the reconfigurability is based on instruction set design, reconfigurable NoC and complete function units as extension to ReRAM array.

II. BACKGROUND

A. ML Algorithms

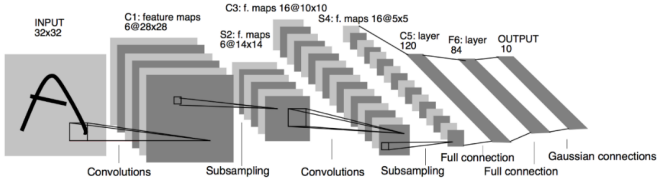

Fig. 1: Lenet 5 architecture [5]

Lenet-5 [5] is a classical and simple deep learning network. It is good for function verification and small-scale benchmark representation. In this project, Lenet-5 was firstly implemented on a circuit level ReRAM array simulator NeuroSim [6] to explore the ReRAM array cell design for matrix computing.

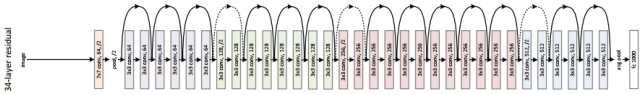

Fig. 2: ResNet34 architecture [7]

ResNet [7] can be thought as the combination of VGG and short-cut path. It enables DNN architecture can be of any depth while not worry about the over-fit problem. At the same time, to guarantee convergence speed, batch normalization layers were introduced to accelerate training speed. From the perspective of this project, ResNet is large and complex enough as the DNN benchmark to verify the performance of this reconfigurable accelerator architecture design.

Algorithm 1: img2col encoding [8]

```
Initialization;
Let N denotes the size of input samples;
Let C denotes the output channel;
Let height/width the height/width of input image;
Let H = (height – K + 2P)/S + 1;
Let W = (width – K + 2P)/S + 1;
Let K the size of kernel ;
for int k = 0; k < N ; k++ do
    for int kernel = 0; output <C; kernel++ do
        for int row = 0; row <H; row++ do
            for int col = 0; col <W ; col++ do
                for int inter = 0; inter <K; inter++
                 do
                    for int inside = 0; inside <K ;
                     inside++ do
                        dtest[k][K^2 * (W * row +
                         col) + inter * K +  inside +
                         output * HWK^2] =
                         dTestInput[k][row *
                         width + col + inside +
                         inter * hight]; end
                    end
                end
            end
        end
    end
end
```

Convolution operation can be slow and cost-expensive without optimization. Irregular memory access pattern, contention and repetitive branch operation can introduce long ideal latency in computing stage. To address these problems, Img2col can pre-process data to machine friendly format via lowering convolution to matrix multiplication making it simple and fast. Considering img2col itself also computing expensive, physical implementation of this algorithm with data-reuse and extra hardware optimizations was taken in this project to avoid performance bottleneck.

B. ReRAM PIM technology

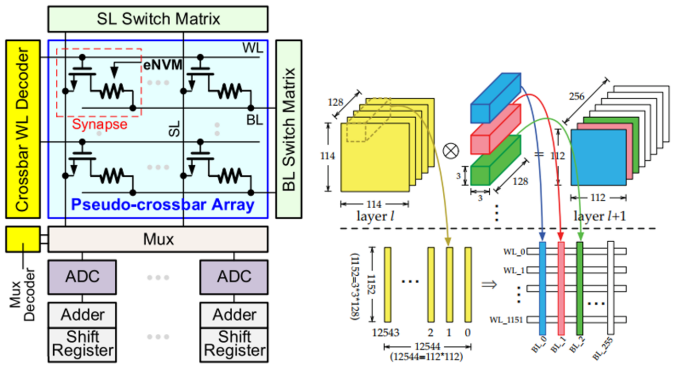

Fig. 3: ReRAM array structure in NeuroSim [6] (left) and inputs and kernel mapping to ReRAM array [12] (right)

When used as computing unit, ReRAM array has very large computing density due to its specialty in doing MAC operations in analogue mode which simply Kirchhoff's law. At the same time, weights can be stored in ReRAM array before computing to minimize the data movement. Due to DNN workloads are computing and data intensive in nature, ReRAM technology suits these purposes well. For the kernel and inputs mapping to ReRAM, the above method from PipeLayer [12] is taken with numbers shown only for indication purpose. This ReRAM mapping method is just the baseline with further optimization methods to solve dimension mismatch problems.

III. Overall Architecture Diagram

To make sure the communication is not the bottleneck of whole performance, the communication network is the integration of 3 sub-networks to guarantee the reconfigurability and cost-efficiency. On-chip buffer is used to receive the input images/weights/instructions from DMA port then organize them to the data format accessible for workload dispatch unit. Another important function of On-chip buffer is to pre-fetch the data for next round in current round to lower the pressure on DMA. The controller is responsible for the overall state control. Workload dispatch unit connects with the root node of the fat-tree topology distribution network which then responsible for delivering the inputs/weights/instructions to each individual PEs for running. There are 256 PEs in total with each PE group consist of 16 PEs. The 2nd component of communication network is the local connection which only connects to adjacent 4 PEs (like mesh topology, will less than 4 in edge nodes) to provide enough bandwidth while as simple as possible. Considering batch communication can be a serious bottleneck in complex network, batch link is implemented with each PE to support broad-cast feature. For down-sampling DNNs, size of results can be quite small compared with input size while the communication pattern can be quite different from the inputs distribution and local communication causing low-utilization of network blocking other communication. So, the third component of the network outputs collection network was introduced to address this problem. This won't cause significant overhead due to down-sampling network normally has much smaller output size. For up-sampling DNN like generative network, the inputs distribution network can be reversed to serve this purpose well.

IV. Instruction Set Architecture

TABLE I: ISA Table of Accelerator

Category	Instruction	Description	Operands
Compute ReRAM$_s$	Array-index/non	Sparsity/Non-Sparsity	R-type, Fusion, Kernel, Input channel, src, des, index
Compute	ALU-Batch	Compute Unit	A-type, batch, size, src, des, forward
NoC-com	NoC-trans	NoC transfer	N-type, Fuse, batch, Kernel, I/O channel, src, des, exit
Control	ACC-Control	General	C-type, IO mode, src, des

The principle of instruction design in this accelerator architecture is to enable reconfigurability, highly efficient communication and coding efficiency. Complete instruction support for all necessary operations is necessary for reconfigurability. As for highly efficient communication, due to

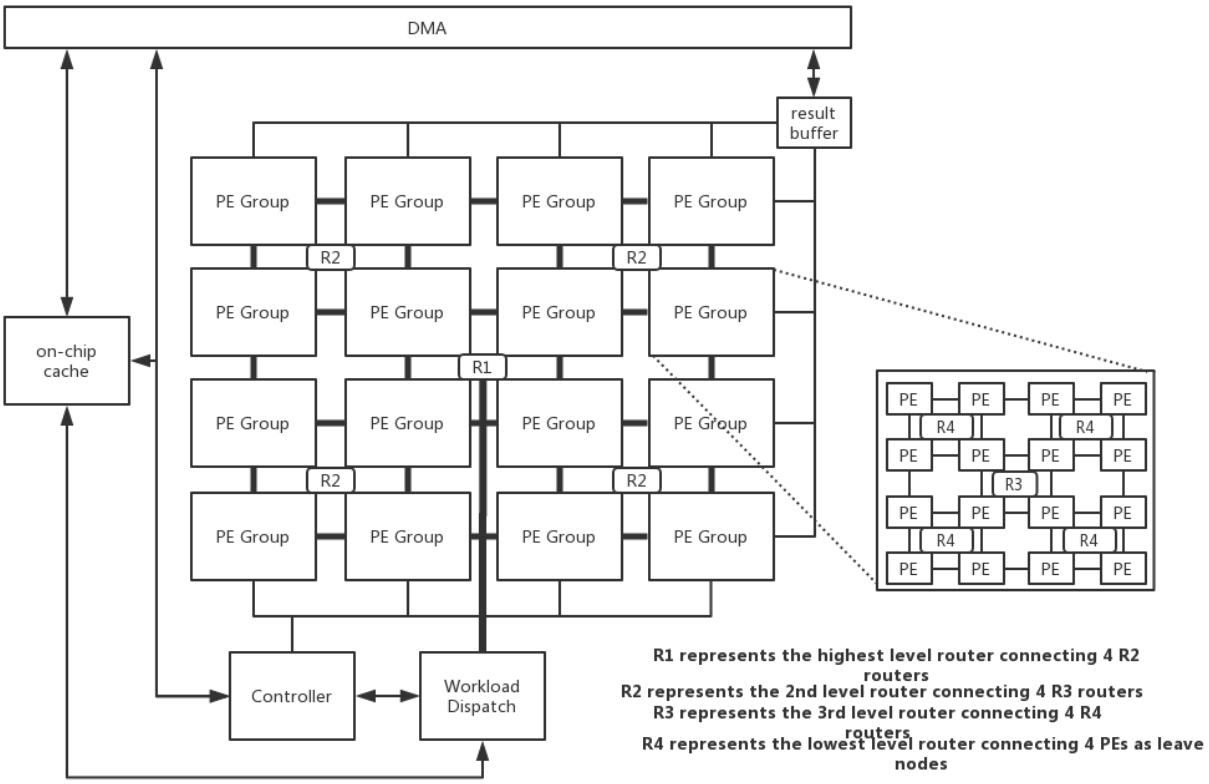

Fig. 4: overall architecture diagram with PE groups

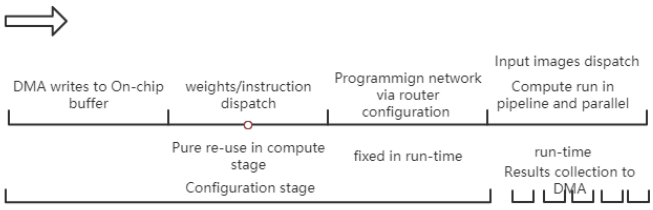

Fig. 5: Accelerator states timing diagram

TABLE II: ISA Extension

R-type	convolution, fully connected
A-type	add, subtract, multiply, divide, inverse square, ReLU, max, min
N-type	distribution, local, collection
C-type	weight load, instruction fetch, data load, workload dispatch, DMA

the control pattern for each PE is well defined in previous of actual computation start and each running actually just repeat the same instructions again for PEs, it is possible to de-couple the instruction from data leaving as much as possible bandwidth for data communication in actual computing phase. This is important for this accelerator architecture because computing abundant architecture mainly bounded by communication bandwidth and utilization efficiency. Most of the instructions will be pre-configured and stored on on-chip PEs. Due to the large variance in these instructions in different category, to minimize the storage overhead, a denser coding was taken introducing 4 different categories. For the programming of PE to different layers in DNN, different instruction blocks can be pre-loaded into the corresponding LUTs of each PE then fetched by the local controller to control the behavior of PE.

V. PE ARCHITECTURE

PE architecture is divided into 2 parts namely ReRAM computing and Arithmetic computing. Different controller for each part for control efficiency. To save on-chip storage space, the space occupied by computation finished inputs is written by the corresponding outputs immediately which requires the size of output smaller than the size of inputs. This is true for down-sample operation while not correct for up-sample which mainly used in GAN network. In the up-sample case, both convolution buffers need to get involved to increase the available space for write operation breaking the pipeline relation between ReRAM computing and Arithmetic computing. In the down-sample case, due to only 1 convolution buffer needed for each computing part, ReRAM and arithmetic computing can happen at the same time to form pipeline. And due to the custom designed

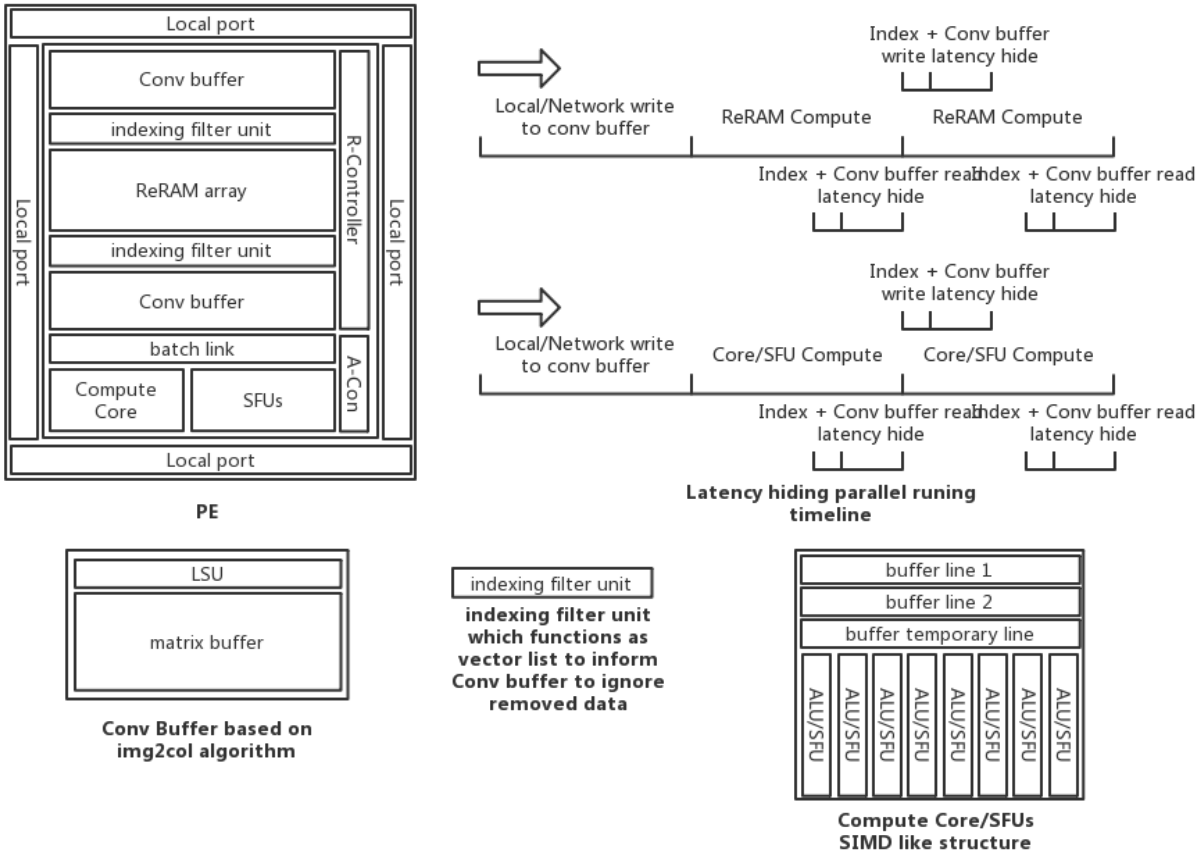

Fig. 6: PE architecture with computing timeline template

convolution buffer and low-overhead indexing filter unit which is modified from another indexing filter unit [9] due to the ReRAM array here can only support 16 columns parallel granularity and the physical implementation is different, the read/write operation can be hidden by the computing stage to achieve high throughput. At the same time, the source of arithmetic/ReRAM computing is switchable to eliminate unnecessary dataflow. When the upper convolution buffer is involving in the arithmetic computing, the new inputs to ReRAM computing can be feed to the lower convolution buffer, then in next round, arithmetic computing source can switch to the lower convolution buffer and ReRAM using the upper one to do transfer and compute. For arithmetic operations, local/network transfer maybe be not needed if no batch operations running simply using the results from ReRAM operations in the same PE. The activation and pooling operations have been integrated into the compute core via resource sharing. Both Compute core and SFU [10] are SIMD-like structure to support multiple operations in parallel.

To simplify the cost of convolution running in hardware, convolution itself can be regarded as the combination of img2col[8] operation and matrix multiplication. The img2col operation can be quite time consuming without special support from hardware because of deep nested loop and rather large storage space. In the worst case, the input size can be expanded to k*k larger (k: kernel size). Which can also mean k*k longer communication latency causing bottleneck in system performance. To remove this bottleneck, the img2col operation and matrix multiplication should be coupled tightly consuming intermediate data quickly and in large bandwidth. One challenge is how to accelerate the img2col operation to match it with the computing power of ReRAM array which can finish 3072 multiply and add operations every cycle in this design. To achieve this, a fast-convolution buffer was designed and implemented in RTL to enable partial inputs re-use and stride parallel read operations to re-organize the input data fast and efficiently. Stride parallel read operation can be regarded as loop unrolling to accelerate img2col in general. The 2nd optimization aims eliminate the workload needed via inputs re-use to minimize the data movement for img2col which can be at k/s scale (s stride) means it

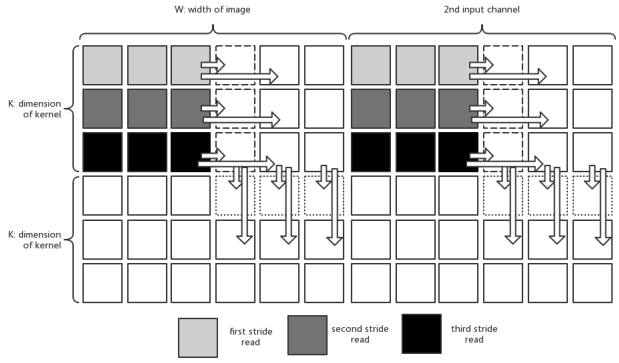

Fig. 7: Convolution buffer sparse/stride access modes

can be quite effective when k»s. At the same time, the fast convolution buffer was integrated with ReRAM array to minimize the communication overhead. ReRAM array can finish the computing from fast-convolution buffer in every 6 cycles. If the read latency and img2col of fast-convolution buffer is lower than 6 cycles, then such latency can be hidden in computing phase making there is no extra img2col overhead but only matrix multiplication latency significantly faster than without customized buffer support.

From the perspective of DNN architecture, the whole architecture can be quite large (3.6E9 parameters for ResNet34 [7]) making it can be hardly mapped to hardware one time completely. If only partial architecture mapped, there will be extra context switch cost, memory access cost and pipeline break. So, pruning to the network can not only bring less workload but also avoid the production of extra cost. However, even the pruning can compress the network to less than 10% of its initial size, the pruned network can be highly irregular requiring complex addressing and control to traverse the whole network which normally induced larger cost offset the benefits of pruning. This problem can be partially solved via a structural pruning method [11] which aims to concentrate dense part together to at least recover some regularity. In this project, a pruning algorithm based on the output channel is taken in which when one output channel is completely pruned, the corresponding input channels will be no longer needed. If the input image is organized as g*C (C input channel, g =k*k), the corresponding columns in this case can be removed easily via columns indexing C. At the same time, if the empty holes left after pruning is still holes, the computing latency on hardware is still not improved due to the structure of parallelism. The indexing filter unit is then introduced to provide skip-like features to replace the pruned parts with next available significant parameters. To do this, indexing filter unit need to feedforward the index number to fast-convolution buffer to override the pruned address with new address. Rather than complex addressing and control for irregular data pattern, only row/column stride index number required to control the sparsity which can be simply done via communication between indexing filter unit and fast-convolution buffer. The speedup can achieve 1/s (s: sparsity degree) in both communication and computing which depends on the pruning effect of DNN and the configuration of hardware. If the batch parallelism number for the ReRAM is set to N, only when N output columns empty at the same time, the corresponding input channels can be ignored.

ReRAM array cell was taken to provide high computing density, low latency and low energy matrix multiplication to solve the limits in computing power. In this project, 1152*256 was taken as the configuration for ReRAM array in one PE. Due to the computing is done in parallel style for both columns and rows of array, a larger size means higher throughput performance. However, the efficiency of mapping can vary for different data size, layer type, DNN architecture and peripheral circuits. The best solution depends on the profiling and statistics of different benchmarks and actual process techniques. For simplicity, 1152*256 was taken with the aim to compare with another accelerator architecture – pipelayer [12] to demonstrate the advantages of this design.

To support a variety of DNN architectures, Compute Core and SFU units are introduced to support pooling, add, subtract, multiply, divide, inverse square root and ReLU operations. The common feature of these operation is either hard or not possible to run on ReRAM array. At the same time, these operations can be reconfigured easily to form different layers like batch normalization layer in DNN to achieve reconfigurability. For complex arithmetic like inverse square root, special function unit common in GPU is responsible for execution to avoid too much iteration needed for simple ALU to improve speed. In order to match the speed of ALU/SFU with the ReRAM array, 32 of each were taken in each PE, this shall not cause strong overhead due to the width of digits is only 4 bits. Higher resolution can be achieved via multiple partial results shift-accumulate to support higher resolution [12].

VI. NoC architecture

There are 3 general communication patterns in this accelerator namely distribution, local and collection. For distribution, weights, instructions, sparsity parameters and input images for each PE are sent from the root node router to each PE. For inference accelerator, the weights, instructions and sparsity parameters can be configured once before inference start then simply repeat until next configuration. On the other hand, the distribution of input images is within the critical path of accelerator. To make sure this will not be a bound to overall performance, the bandwidth needed for each PE is calculated based on the data consumption rate of ReRAM array. For local communication, this can happen when the results of PE are not collected directly to result buffer but transferred to next PE for further processing. Possible cases can be when the input channels are larger than 128, then multiple PEs need to accumulate the partial results to generate the full result. Or next PE corresponding to the next layer in DNN architecture. In these cases, using

the distribution network can be less efficient due to communication via routers can cause conflicts. One optimization in this architecture is to make sure the relevant PEs can be as close as possible to minimize the communication overhead that PEs can be connected to nearby PEs directly. As for collection pattern, the output of DNN architecture is always much smaller than the input size. So, a customized collection network can transfer the results more efficient lowering the pressure on distribution network.

A. Communication pattern

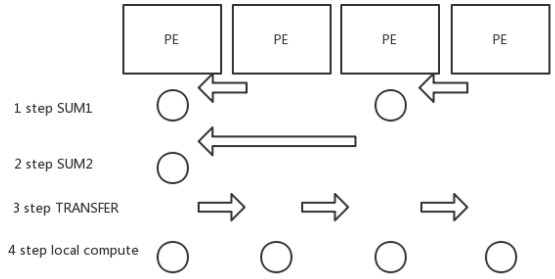

Fig. 8: Local communication pattern for BN to achieve scalable computation utilization

One representative batch operation is bath normalization which requires complex communication pattern involve broad-cast features. The batch link inside PE can support this well. In figure 8, it is shown how the communication pattern can be combined with distributive computing to achieve scalable solution. The naïve method is based on the serial accumulation process and hop by hop to get the average statistics which will result O(n) time complexity as workload grow. In the new distributive method, the latency only grows O(logn) to make all batch operations scalable on the network.

B. Network components

A standard router microarchitecture can include BW (buffer write), RC (route compute), VA (VC allocation), SA (switch allocation), BR (buffer read). ST (switch traversal), LT (link traversal) 7 stages in total. However, this is not needed in NoC for DNN accelerator architecture because the communication pattern between each PE is well defined before runtime. To simplify the microarchitecture of router, input buffers, route compute unit and VC allocator can be removed to save power and area of router. On the other hand, to improve the multi-cast capability, crossbar switch can be fourfold to adapt to the fat tree topology of distribution network better to enable cast to 4 next level routers at the same time. Another contribution in this project is the route computing has been replaced with route encoding that which packet should go which PE/router has been encoded in the structure of distribution network that can split automatically in each hard-wired divergence structure of router only need to specify the entrance point in workload dispatch unit. For this distribution network to support local communication where the source of communication maybe not from workload dispatch unit but from PEs and collection for up-sampling network, the link is bi-directional with SW Allocator to program the connection pattern in runtime. Due to the SW allocator can pre-configure the distribution path in initialization stage of accelerator, the only overhead left for router is simply ST and LT stages. Rather than use expensive virtual channels [13] , pipeline, bypass and arbitration, the router can be optimized just by the well-defined communication patterns of DNN itself.

VII. Workload mapping and resource allocation optimization

A. scalable mapping framework

For spatial architecture, the workload mapping is critical for computing efficiency because of the relative location between PEs has significant influence on communication pattern. At the same time, good resource allocation can eliminate potential performance bottlenecks in limited resource budget. Here, 2 mapping algorithms were provided to solve this problem. The selection of mapping depends on analysis of workload. The main different between them is whether the global fat-tree network got involved in the inter-layer communication or not. The first mapping method aims to strength the local communication within the same layer group to avoid extra communication path when distributed separately. While, the disadvantage is inefficient utilization of bandwidth in the network due to linkage between layers in DNN normally dominated by one way while mesh network is 2-dimension network. Though we can use intra-layer as another dimension to fully utilize bandwidth within the same layer group. There can still left 1 dimension not utilized efficient when different layer groups connect. Considering the inputs are re-used when propagate alone the network, only small part of the global fat-tree network used for inputs distribution in small batch number. The left part can be taken to facilitate more complex communication patterns like short-cut (cross layer) [7] to improve pipeline performance. As for the global folding mapping, the algorithm works via compress 2-dimension layer group to 1 dimension to maximize the bandwidth utilization. The effect is individual layer group can be quite long that unable to fit into the square array of PEs. So, it then needs to be folded. The core of folding here is to fold the neighbor layers groups together so the individual layer group is not connected like local mapping but distributed separately. In this case, different group layers are connected set by set. To avoid extra communication distance and bandwidth contention, global fat-tree need to function as inter-layer communication to connect the far parts together. On the contrary, the global fat-tree network need time-multiplexing to provide support for special communication pattern which may lower the pipeline performance. Another negative case can be the network is too small that parallelism is more important that too much inputs distribution causing there not enough space for inter-layer communication.

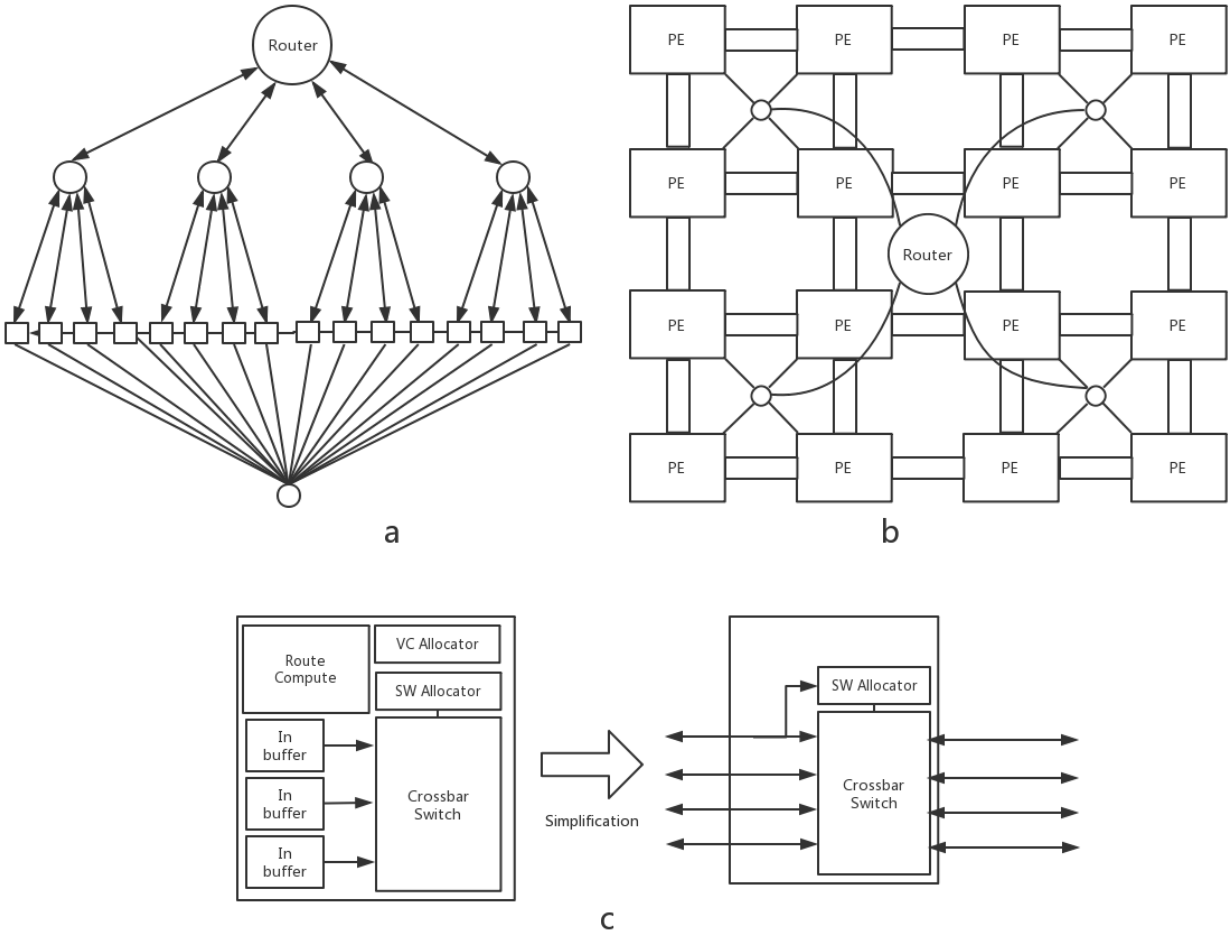

Fig. 9: a: topology of NoC, b: 2D expansion of Noc, c: NoC router Microarchitecture vs traditional router design

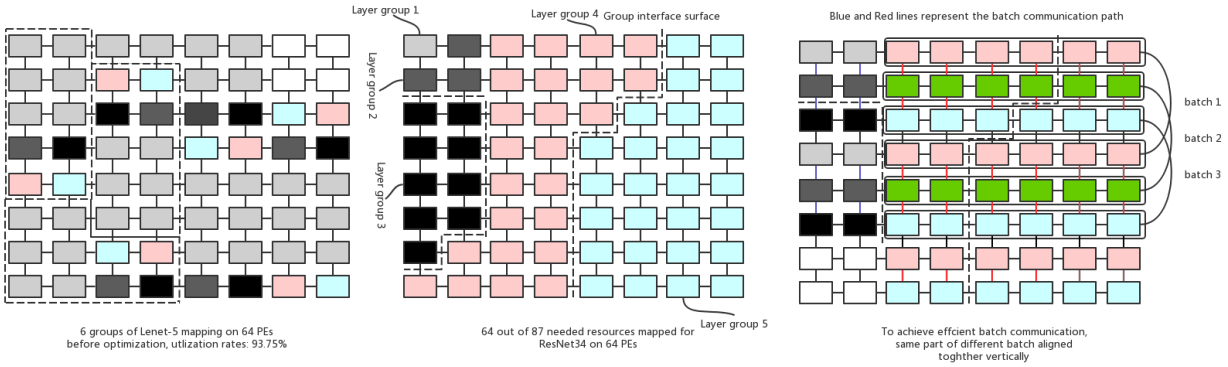

Fig. 10: scalable and efficient mapping on PE array for different size DNN

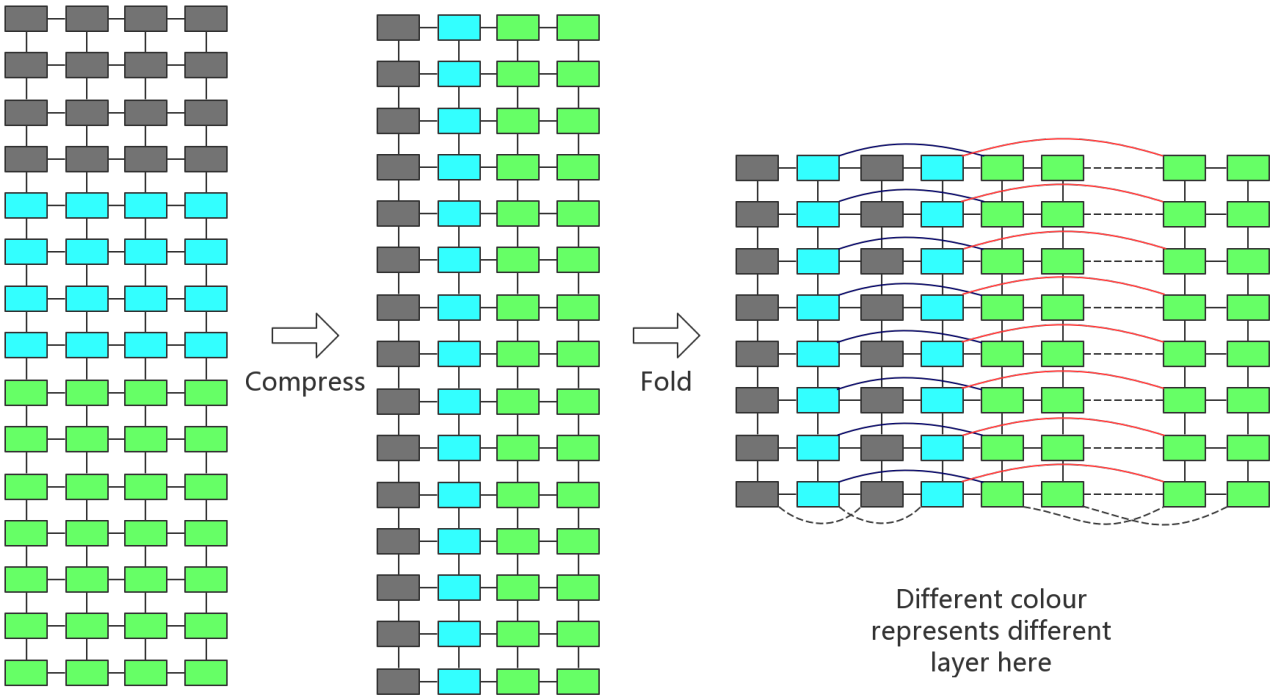

Fig. 11: global folding mapping to maximize inter-layer bandwidth utilization

TABLE III: Duty partition of two mapping strategies

Local Mapping	
Local connection	Usages: Inter-layer, intra-layer, batch
Global connection	Usages: inputs/weights/instructions distribution, cross-layer
Global folding mapping	
Local connection	Usages: Inter-layer, intra-layer, batch
Global connection	Usages: inputs/weights/instructions distribution, inter-layer, cross-layer

B. Potential mapping cases

- (small network, small input image, high batch number) multiple groups of full network run in parallel (multiple input images finished when go through the chip) .
- multiple groups of full network run in parallel (multiple input images partial finished when go through the chip).
- multiple groups of partial network run in parallel (multiple input images finished when go through the chip).
- multiple groups of partial network run in parallel (multiple input images partial finished when go through the chip).
- (large network, small input image) single full network run at a time (multiple input images finished when go through the chip).
- single full network run at a time (multiple input partial images finished when go through the chip).
- single full network run at a time (single input images finished when go through the chip).
- single full network run at a time (single input images partial finished when go through the chip).
- single partial network run at a time (single input images finished when go through the chip).
- (very large network, very large input image, low batch number) single partial network run at a time (single input images partial finished when go through the chip).

C. shortest path mapping algorithm for PEs

The mapping of workload should reflect the degree of relevance to minimize the global communication cost. The degree of relevance is represented as a combination of layer internal, inter layer and batch equivalence.

Algorithm 2: Relevance ranking based workload mapping

```
Let Li denote the ith layer of DNN;
Let Pm denote the mth member of batch;
Let An denote the nth element of layer;
Let Gk denote the minimum PE set in the NoC;
while Li != last layer in DNN do
    for each observed entry Pm in Bi do
        for each observed entry An in Pm do
            if Gk is full then
                k = k+ 1 ;
                Mapping An to Gk;
            else
                Mapping An to Gk;
            end
        end
    end
    i= i+ 1;
end
```

D. Efficient mapping diagram for ReRAM array inside PE

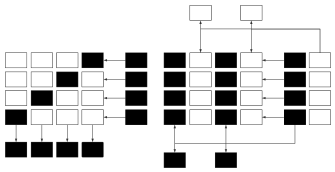

Fig. 12: Efficient ReRAM array mapping when abundant row/column and only column abundant

It is important to merge or divide different data formats to match with the dimension of ReRAM array to achieve high efficiency. In the left graph of figure 12, four sets of inputs merged together to share the same array with four sets of outputs to increase the efficiency 4 times. In the right graph of figure 12, the inputs time-multiplexing the same array with care for the column multiplexing granularity.

E. mapping optimization for throughput

Due to the latency in-balance of each stage, it is possible to re-allocate resources for specific performance bottlenecks within resource constraints to convert parallelism to pipeline to achieve overall system performance improvement.

Algorithm 3: in-demand resource re-allocate mapping optimization

```
Let Li denote the ith layer of DNN;
Let Tm denote the inference latency;
Let Algo3 be the algorithm 3 for recursive purpose;
Let Sn be the overall workload mapping settings for
  inference latency calculation;
Let Gk denote the group of layers in DNN;
T1 = func(S0) % latency function for mapping S0;
T2 = T1 ;
while T2<=T1 do
    for each Li in G0 do
        G1 = G0 – Li;
        Do Algo3 on G1 with Li optimized ;
        T3 = func(S1) with S1 convert from S0 ;
        Do Algo3 on G1 with Li non-optimized ;
        T4 = func(S2) with S2 convert from S0 ;
        if T3<T4 then
            T5 = T3 ;
            if T5 < T2 then
                T2 = T5;
                Update(S1) ;
            end
        else
            T5 = T4 ;
            if T5 < T2 then
                T2 = T5;
                Update(S2) ;
            end
        end
    end
end
```

VIII. ReRAM array for CNN function validation

A. validation flow

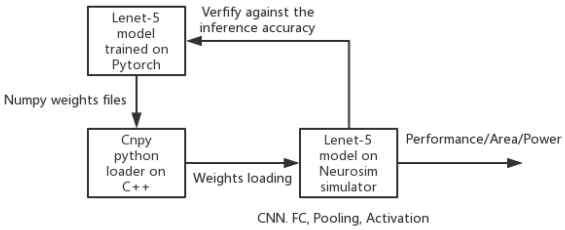

Fig. 13: validation model for Lenet-5 via NeuroSim [6] C++ model and Pytorch [14]

This step is to verify whether DNN can be mapped to ReRAM array correctly and efficiently without expensive pre-process. For ease of verification and building, Lenet-5 was implemented on ReRAM array circuit level simulator for power/area/performance test.

B. re-structure the input image to deploy img2col

img2col algorithm was implemented in the simulator to convert convolution to matrix multiplication to achieve fast convolution operation.

C. Error resilient techniques and low precision representation

Because ReRAM array is non-linear device and there also exists bias for DAC and ADC in peripheral circuits, the weights trained from Lenet-5 model in Pytorch [14] can't be used directly on ReRAM Array. In analysis to the intermediate results traverse each layer in Lenet-5 network, the divergence is growing. In the initial stage of this project, training is also considered part of the accelerator architecture. The basic idea is to train the weights on accelerator directly to avoid such mismatch while not successful. In the end, the weights trained in pytorch were reverse transformed via the statistical non-linear property of ReRAM so after the non-linear physical property effects of ReRAM the original weights can be recovered. As for the bias and structure errors from circuits, a feedback loop was introduced to offset these disturbances. The results show the accuracy of inference on accelerator is identical to that on Pytorch platform [14].

There are multiple papers indicate high precision numbers maybe not necessary for DNN accelerator [15]. After experiment on Lenet-5 model with double precision reference: 95.7%, It shows 1bit: 42.1% 2bit: 92.8% 3bit: 94.8% 4bit: 95.4% 5 bit: 95.6% 6bit: 95.6% 8bit: 95.7%. There is only 0.3% accuracy loss for 4bit representation while the data has been compressed to only 12.5% (compared with double data type). The 4bits setting is taken here for ReRAM array cell.

IX. Training flow

The training flow is not taken on the accelerator architecture in the end due to not enough time to verify its generality and do the optimization. The feedforward phase of training is the same as inference with intermediate inputs need to be recorded for delta update on weights. For the backpropagate phase, there are up-sample and correlation transformations to backpropagate the errors from the last layer [5]. To accelerate the convergence of DNN, optimizer is also required to control the learning process dynamically. It's firstly tried to implement the training on Lenet-5 ReRAM array to lean the non-linear physical property of ReRAM while failed due to low precision after training. Considering the training algorithm is still evolving, using the DNN framework like Pytorch/Tensorflow is better choice, the training is done on Pytorch later.

X. Validation methodology

A. Performance modeling

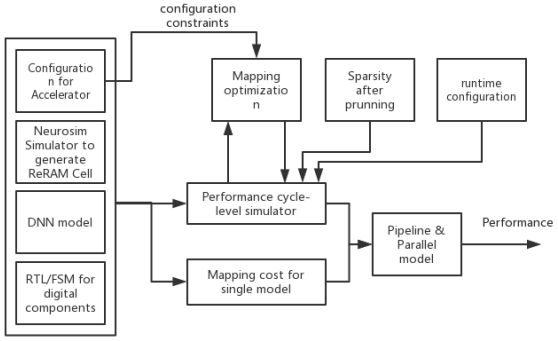

Fig. 14: performance modeling framework

B. Platform setting and accelerator specification

TABLE IV: Evaluation Platform

Type	Platform Characteristics
CPU	Intel Xeon E5-2670v3, 10-cores per socket, Dual Socket, 128GB DDR4
GPU	Nvidia Tesla P100, 3584 CUDA Cores, 16GB HBM2 with CuDNN5.0

A cycle-level performance simulator of the accelerator architecture based on Python has been implemented to validate its performance. The design of this performance simulator can be thought as the integration of timing models of each sub-components plus some compiler-like optimizations. There are different sources of the timing parameters used in this simulator. For on-chip cache, destiny cache simulator [17] can generate read/writer latency for given cache configuration. For ReRAM array, Neurosim was used to generate the timing model of ReRAM array. For the left digital components, the timing parameters are from the corresponding RTL and FSM diagram. Multiple DNN benchmark can then be constructed on these sub-components models to form new performance model corresponding to the given DNN. In this project, 2 examples namely Lenet-5 and ResNet34 have been implemented and tested. As for the area and power parameters, NanGate FreePDK90nm technology was used to synthesis part of RTL code (fast-convolution buffer) in Virtuoso Encounter to get the layout area and power estimates. For fairness, the area and power estimates are then scaled to 22nm technology to reflect coherence in the whole design. Then destiny cache simulator and Neurosim were used to get the area and power parameters for the corresponding parts.

TABLE V: Specification for Reconfigurable ReRAM based accelerator design

PE specification	
ReRAM Array	1152*256 dimension, 4bits precision, numColMuxed = 16
Conv-buffer	84KB*2 6x128x224 dimension
SFU	32 units. transcendental functions, Inverse square root 8 cycles
ALU	32 units, Max, min, ReLU, add, subtract, multiply, divide 1 or 4 cycles
On-chip Cache	
Cache	4MB, 4096 bits wordline, 8 set associative, Destiny on Chip Cache simulator
NoC Specification	
Topology	Fat tree topology 1-4-16-64, Local Mesh connect, collection 256-1 network
Router	5 radix, 1 cycle, 1-4 Multicast, None, 4-1 collection
Link	16bits, 1 cycle link,
Node	256 PEs
DMA	
Hypertransport	Off-chip Network, 6.4GB/s,

TABLE VI: DNN Benchmarks specifications

DNN benchmarksn					
Algorithms	Data structure	Dataset	Weights	FLOPS	Layer type
ResNet34	224x224x3x4	ImageNet	21.282M	3.6e9	CNN, BN, ReLU, FC
Lenet-5	20x20x4	MNIST	60K	3.4e5	CNN, ReLU, FC

As for the comparison with state-of-art sever processors, Nvidia P100 [17] was chosen as the GPU and Xeon E5-2670 was chosen as the CPU. Due to there is OS workload on CPU, the power profiling maybe not accurate enough, only GPU was profiled in nvidia-smi tool to get the runtime power. For the performance results, Pytorch 1.1 was used to run ML benchmark on CPU and GPU. The GPU is also accelerated with NVIDIA cuDNN library.

XI. Evaluation

A. Summary of results

Convolution buffer has been implemented in RTL for Lenet-5 for verification purpose while not implemented for the final large accelerator design due to the university encounter license that I have only allow for small scale design. The final size for convolution buffer is at the size of 224x128x6 which is too large for the license.

TABLE VII: Performance

Performance Comparison					
Algorithms	Workload Size	Re-ReRAM Latency	Nvidia-P100 Latency	Xeon E5-2670 Latency	Re-ReRAM(FP4) Vs GPU(32)
ResNet34	10000	0.019ms	3.500ms	40.000ms	184.21X (Norm 23.03X)
Lenet-5	1	0.023ms	2.500ms	1530.00ms	108.70X (Norm 13.59X)

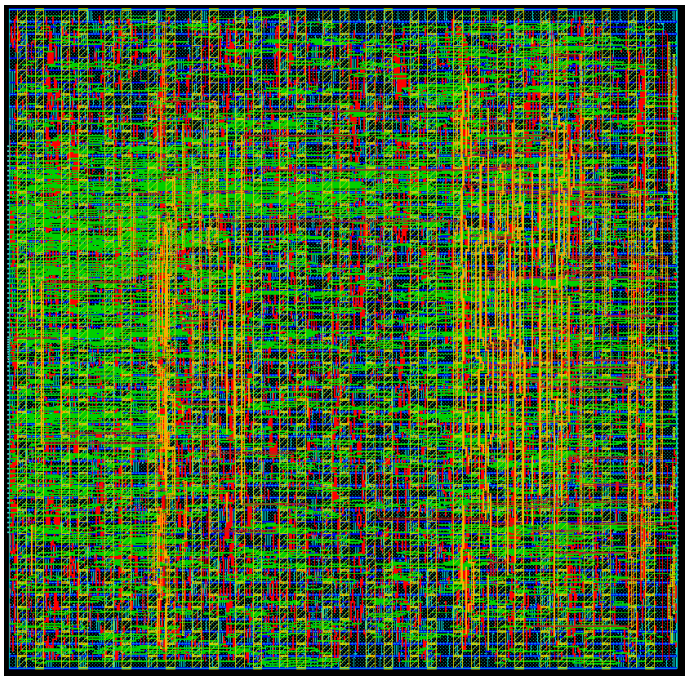

Fig. 15: the layout of 20x6 convolution buffer with area 0.04mm$^2 and 1GHz power 1.975mW$

TABLE VIII: physical parameters from simulators

NeuroSim2.0 ReRAM array parametersn	
Matrix Array 1152x256	Compute rates for colmux: 5.77ns Total area=7.7373e-09 m^2, $power 0.115mW$
Destiny cache simulator on-chip cache parameters	
2D$_e DRAM$	Total Area = 4.404mm^2, $cacheread$: $0.605nscachewrite$: $0.505power 1.13W$

B. Performance bound analysis with implemented cycle-level simulator

The initial results after mapping is at least 87 PEs needed to map the ResNet34 to on chip with maximum inter-layer pipeline stage near 200000 cycles. After the optimization runs the 1st iteration, the longest 8th layer can be minimized to 110000 cycles with extra 1 PEs added (compute latency decreased to half while transfer bandwidth stay the same due to upper-level producer does not change). The relative speedup is 74.00% for 1st optimization iteration. After the 2nd iteration, the relative speedup is 59.50% with extra 8 PEs added. Further optimization process stops due to the cost is growing exponentially and can't offset the drawbacks anymore.

The cycle of inter-layer pipeline stage depends on the slowest stage. To optimize the general speed, the slowest stage should be accelerated with reasonable resources. The bandwidth depends on the mapping and physical parameters of the accelerator. Unless optimized in mapping algorithm, it is hard to change it in run-time. In general, the accelerator architecture normally bound by communication bandwidth and ReRAM computing part can be easily accelerated via allocating more PEs. At the same time, more PEs allocated to one model also means the decrease in parallelism. It is the tradeoff between these 2 factors to achieve better

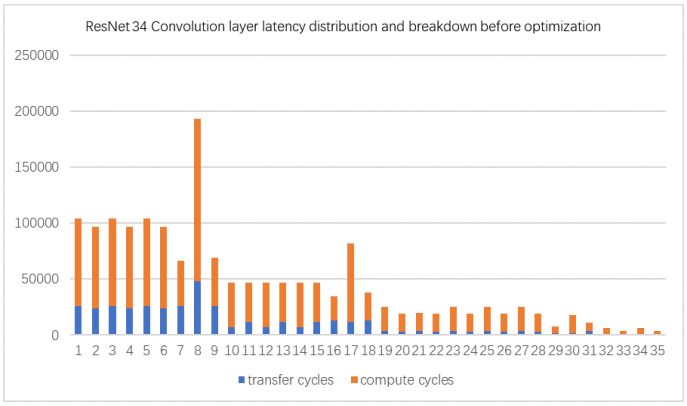

Fig. 16: ResNet34 convolution layer latency distribution and breakdown before optimization

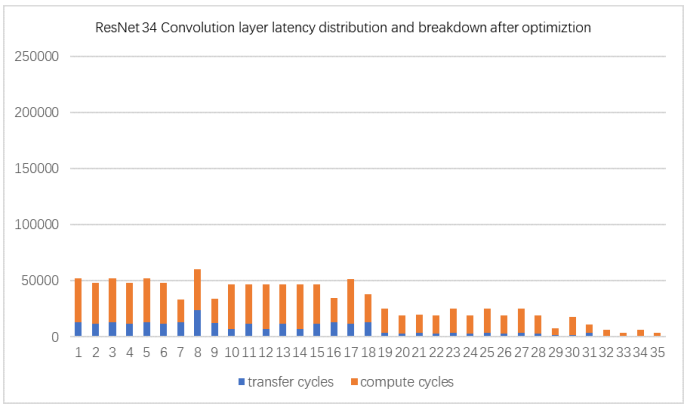

Fig. 17: ResNet34 convolution layer latency distribution and breakdown after optimization

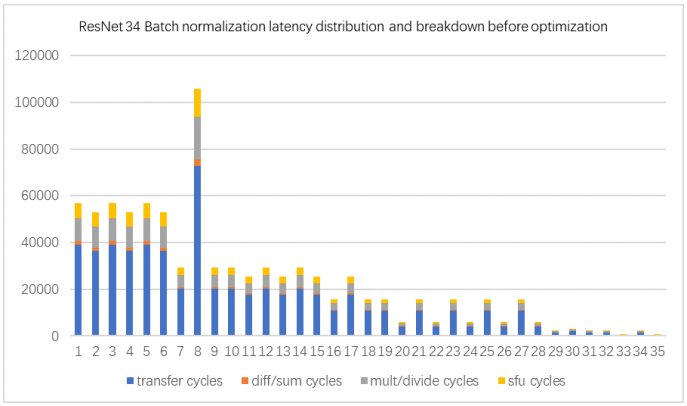

Fig. 18: ResNet34 Batch Normalization latency distribution and breakdown before optimization

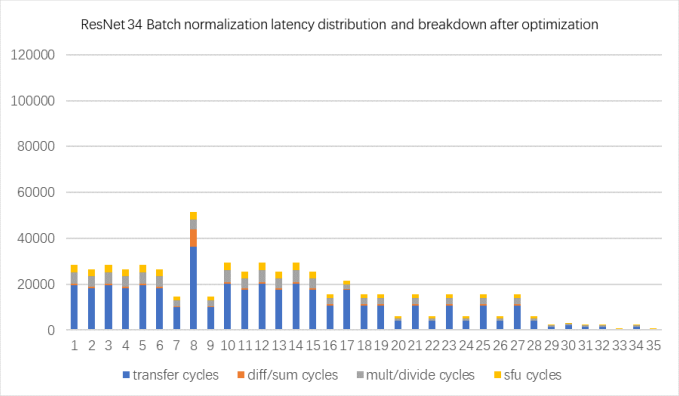

Fig. 19: ResNet34 Batch Normalization latency distribution and breakdown after optimization

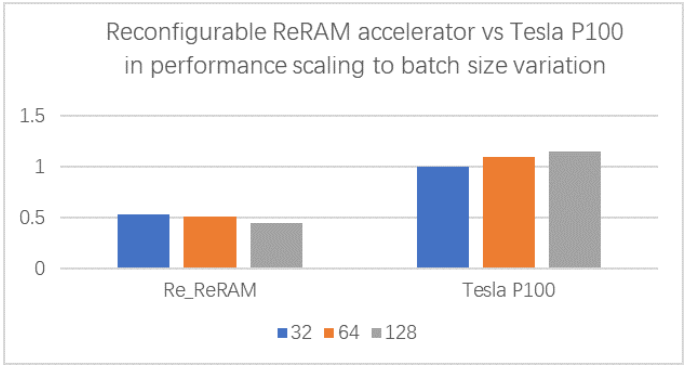

Fig. 21: Re-ReRAM Vs Tesla P100 Performance scaling in batch mode

performance.

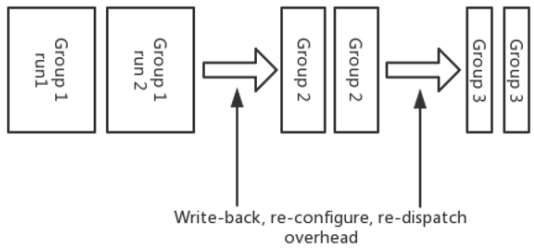

Fig. 20: potential separation running diagram to avoid inter-layer in-balance

Considering the exponential variances exist for different layers, such in-demand resource re-allocate method still can't remove such in-balance completely. One potential method is to partition running stages separately with only one-layer type or part of the layer types at the same time. The whole chip resources can then be fully utilized to parallel this layer or layer group with latency specific to this layer or layer group. It is un-necessary to synchronize with other layer type so no idle cycle for fast stages in this case. The results are then write-back to on-chip cache ready for dispatch for next round rather than transfer to next layer via local links. The chip is then re-configured for the next layer. Such re-configuration can be slow causing significant overhead. It can be necessary to apportion such overhead with enough running times. Basically, this method is only suitable for part of the DNN workload with strong dependence on DNN architecture, communication patterns, on-chip resources and application purpose.

GPU can benefit from partials results re-use which means it can be faster as the batch size increase while ReRAM can only benefit from inputs re-use. For larger batch size, ReRAM accelerator suffer more from limited bandwidth in batch communication. On the other hand, GPU use hierarchical distributed cache system to communicate with higher power consumption.

XII. Related Works

A. ReRAM based DNN accelerators

Pipelayer[12] is a ReRAM based DNN accelerator which has the same overall architecture of PRIME[19] with extra support to training acceleration and intra- and inter-layer pipeline. At the same time, they used a spike-based data input and output to eliminate the overhead from existing voltage-evel based scheme. When compared with Re-ReRAM, there is no support for batch communication based on reconfigurable communication network and extra arithmetic units.

B. NoC Design in DNN accelerators

Eyeriss[20] used separate buses for scatters and gathers operations. Diannao [21] and Shidiannao[22] used mesh interconnects for data communications. Maeri[1] used a new tree topology called Augmented Reduction Tree (ART) which is based on binary tree with additional links to adapt to the varying data-flow patterns in DNN. One unique contribution is this ART can support sparse convolution operation via configurable Virtual Neuro Construction and sparse mapping. Due to different pruning strategy, the sparse network on Re-ReRAM is carried on indexing filter unit.

XIII. Conclusion

In general, Lenet-5 is a representative small-scale ML model which can run in a large batch mode to earn benefits from weights and inputs re-use. ResNet34 is a representative large-scale ML model which need much more computing resource to support large batch mode. Via testing on these 2 representative models, the adaptivity of the accelerator to general set of ML models can be verified. The size of ML model is an important factor in the mapping efficiency of the designed accelerator architecture. At the same time, this is also a main drawback of most ReRAM based accelerator that the architecture can't adapt to the size of ML model efficiently. The ReRAM array normally has high row number to increase the multiply-add operations parallelism to offset the overhead from peripheral circuits. However, if the size of model is not big enough, some computing points in array just replaced with 0 value meaning no real computing happened while still occupying the computing

resources. Multiple inputs coalescing optimization has been taken to increase the mapping efficiency while this problem can hardly be solved completely due to the structure of ReRAM array. On the other hand, digital computing units on GPU are fully configurable to match with the size of different ML models efficiently. After the analysis to the breakdown of performance statistics on ResNet34, it is found that Batch normalization (BN) [18] is mainly bound by communication speed of NoC and CNN mainly bound by computing resource. Computing resource can be reconfigured easily by duplicating the number of PEs allocated. While, the reconfiguration to the NoC communication bandwidth is difficult because reconfiguration can only control the direction and transfer pattern while can't re-allocate the fixed bandwidth. In the end, the pipeline stage latency depends on the worst case of both BN and CNN with CNN still has room to speedup while BN bound by communication bandwidth. So, the overall architecture is bound by the communication rate. One potential solution can be introducing extra switchable links like FPGA to achieve deeper reconfigurability or allocating extra link bandwidth for few critical paths which needs statistical data for most DNN benchmarks. One challenge encountered is each PE has private ports to distribution network to guarantee adaptivity in case of small DNN Models while may under utlization for larger DNN models because large models normally heavy in local communication due to very long processing stages while light in input stages. On the other hand, GPU can match the communication and compute well while bound by not enough computing resources. The key is how to achieve both compute and communication high speed enough and both reconfigurable to achieve balance. The challenge can be summarized as ReRAM array is more novel than NoC which may need further development.

REFERENCES

[1] H. Kwon,, A. Samajdar and T. Krishna, "A Communication-Centric Approach for Designing Flexible DNN Accelerators," in IEEE Micro, vol. 38, no. 6, pp. 25-35, 1 Nov.-Dec. 2018.

[2] A. Shafiee et al., "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in Proc. ISCA, 2016.

[3] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017. doi: 10.1109/JPROC.2017.2761740

[4] T. Nowatzki, V. Gangadhar, K. Sankaralingam and G. Wright, "Domain Specialization Is Generally Unnecessary for Accelerators," in IEEE Micro, vol. 37, no. 3, pp. 40-50, 2017.

[5] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[6] P. Chen, X. Peng and S. Yu, "NeuroSim: A Circuit-Level Macro Model for Benchmarking Neuro-Inspired Architectures in Online Learning," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 12, pp. 3067-3080, Dec. 2018.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, 2016, pp. 770-778.

[8] K. Chellapilla, S. Puri, and P. Simard, "High performance convolutional neural networks for document processing," in 10th International Workshop on Frontiers in Handwriting Recognition. Suvisoft, 2006.

[9] P. Wang, Y. Ji, C. Hong, Y. Lyu, D. Wang and Y. Xie, "SNrram: An Efficient Sparse Neural Network Computation Architecture Based on Resistive Random-Access Memory," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, 2018, pp. 1-6.

[10] Z. Jia, M. Maggioni, J. Smith, D. P. Scarpazza. (2019). "Dissecting the NVidia Turing T4 GPU via Microbenchmarking." [Online]. Available: https://arxiv.org/abs/1903.07486

[11] W. Wen et al. Learning structured sparsity in deep neural networks. In NIPS, 2016.

[12] L. Song, X. Qian, H. Li and Y. Chen, "PipeLayer: A Pipelined ReRAM-Based Accelerator for Deep Learning," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 541-552.

[13] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif and C. R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," 2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06), Orlando, FL, 2006, pp. 333-346.

[14] A. Paszke et al., "Automatic differentiation in pytorch," in Proc. NIPS. Located Long Beach Conv. Entertainment Center, Long Beach, CA, USA, 2017.

[15] S. Han et al., "EIE: Efficient Inference Engine on Compressed Deep Neural Network," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 243-254.

[16] M. Poremba, S. Mittal, D. Li, J. S. Vetter and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," 2015 Design, Automation Test in Europe Conference Exhibition (DATE), Grenoble, 2015, pp. 1543-1546.

[17] NVIDIA Tesla P100 White paper, NVIDIA, 2016, Open Access: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[18] Sergey Ioffe et al. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In International conference on machine learning (ICML). 448–456.

[19] P. Chi et al., "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," in Proc. ISCA, 2016.

[20] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energyefficient reconfigurable accelerator for deep convolutional neural networks," IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127–138,2017.

[21] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in ASPLOS, pp. 269–284, 2014.

[22] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen,and O. Temam, "Shidiannao: Shifting vision processing closer to the sensor," in ACM SIGARCH Computer Architecture News, vol. 43, pp. 92–104, ACM, 2015