STIFT: A Spatio-Temporal Integrated Folding Tree for Efficient Reductions in Flexible DNN Accelerators

FRANCISCO MUÑOZ-MARTÍNEZ, Universidad de Murcia (Spain)
JOSÉ L. ABELLÁN, Universidad Católica de Murcia (Spain)
MANUEL E. ACACIO, Universidad de Murcia (Spain)
TUSHAR KRISHNA, Georgia Institute of Technology (USA)

Increasing deployment of Deep Neural Networks (DNNs) recently fueled interest in the development of specific accelerator architectures capable of meeting their stringent performance and energy consumption requirements. DNN accelerators can be organized around three separate NoCs, namely distribution, multiplier and reduction networks (or DN, MN and RN, respectively) between the global buffer(s) and the compute units (multipliers/adders). Among them, the RN, used to generate and reduce the partial sums produced during DNN processing, is a first-order driver of the area and energy efficiency of the accelerator. RNs can be orchestrated to exploit a Temporal, Spatial or Spatio-Temporal reduction dataflow. Among these, Spatio-Temporal reduction is the one that has shown superior performance. However, as we demonstrate in this work, a state-of-the-art implementation of the Spatio-Temporal reduction dataflow, based on the addition of Accumulators (Ac) to the RN (i.e. RN+Ac strategy), can result into significant area and energy expenses. To cope with this important issue, we propose STIFT (that stands for *Spatio-Temporal Integrated Folding Tree*) that implements the Spatio-Temporal reduction dataflow entirely on the RN hardware substrate (i.e. without the need of the extra accumulators). STIFT results into significant area and power savings regarding the more complex RN+Ac strategy, at the same time its performance advantage is preserved.

CCS Concepts: • **Hardware → Hardware accelerators**; **Emerging architectures**.

Additional Key Words and Phrases: Deep Neural Networks, DNN Accelerators, Computer Architecture, Networks-On-Chip

1 INTRODUCTION

Deep Neural Networks (DNNs) have recently revolutionized many well-known domains, such as computer vision, language translation, or audio recognition, just to mention a few [31].

The execution of a DNN model comprises two different processing phases, namely training and inference. Training is usually a long off-line process that is accelerated via powerful data centers with heterogeneous (CPU+GPU/FPGA) server nodes. As a result of this phase, the DNN model weights are established so that it is then ready for deployment and usage. The usage of a trained DNN model is called the inference procedure (a.k.a., prediction) and is typically run on an edge-computing device featuring stringent computing power and energy requirements [31].

The computation of the dot products entailed by the DNN inference phase requires the execution of simple Multiply-and-Accumulation operations (i.e. MACs), albeit in a very large quantity (e.g. 15.5 billion MACs approximately for the

Authors' addresses: Francisco Muñoz-Martínez, Universidad de Murcia (Spain), francisco.munoz2@um.es; José L. Abellán, Universidad Católica de Murcia (Spain), jlabellan@ucam.edu; Manuel E. Acacio, Universidad de Murcia (Spain), meacacio@um.es; Tushar Krishna, Georgia Institute of Technology (USA), tushar@ece.gatech.edu.

Manuscript submitted to ACM

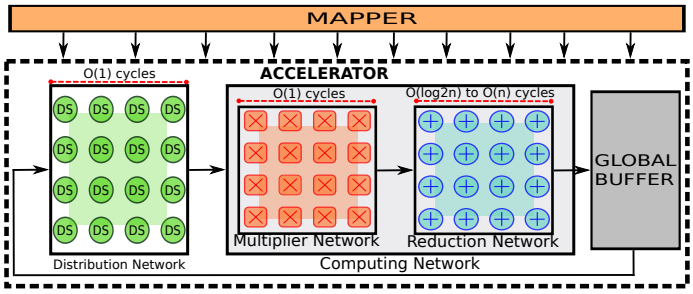

Fig. 1. Conceptual view of a typical DL accelerator.

VGG16 DNN model [30]). In order to fulfill the performance and energy consumption requirements of the inference procedure, the research community has embarked on the mass development of specialized DNN accelerator architectures [9, 11, 13, 18, 23, 26, 29]. These proposals are usually built using a large number of connected multiplier and adder units. The adder units massively reduce the large number of partial sums (psums) generated by the multiplication operations.

Prior to execution of the inference procedure on a DNN accelerator, a mapper [10, 32] (see Figure 1) selects, based on the hardware capabilities and the specific characteristics of the workload (e.g., type and dimensions of the DNN layers), the number of dot products that will be calculated in parallel, and the group of multipliers and adders in charge of computing each dot product. In this work, we will use the term *cluster* to refer to a group of multiplier units in charge of a certain dot product. Usually, the number of multipliers in the cluster is smaller than the number of products required for the assigned dot product, so that several iterations are required. The result of each iteration is accumulated with that of the subsequent one, and so on, until the full dot product is completed. This process is known as *folding* [10].

Once the hardware configuration is determined by the mapper, the processing of each of the clusters in most recent proposals usually relies on a hardware implementing a 3-stage pipeline based on three network fabrics [19] (see Figure 1): the Distribution Network (DN), a global network that connects the global scratchpad SRAM (i.e., the Global Buffer) to the multipliers, is in charge of distributing all the operands for each cluster; the Multiplier Network (MN), a local network between the multipliers that enables reuse of weights (or inputs), performs the multiplication operations; and the Reduction Network (RN), a global network connecting the MN and the Global Buffer, carries out all the accumulations needed for the reduction phase. The MN and RN contain all the compute units (multipliers and adders, respectively) and along with the DN, define all the permitted mappings, and thus, determine the *dataflow* [21, 31] and the energy efficiency of the execution. These NoC-based designs have been shown capable of reducing the expensive off-chip memory accesses, and thus, to improve performance and energy consumption [8].

The separation between the DN, MN and RN can be physical (e.g. MAERI [23], SIGMA [29]) or logical (e.g., Eyerissv2 [11], Google's TPU [18]). We focus on the first type of accelerator architectures (*flexible* DNN accelerators), as they have been shown to provide increased flexibility and better performance than the second type (*rigid* accelerators) [23, 29].

The DN, MN and RN in these flexible accelerator designs are typically implemented using lightweight (energy- and area-efficient) microswitches [8, 22], which enable independent reconfiguration of each on-chip network in order to efficiently map different dot product partitions through the creation of dynamic-size clusters enabling single-cycle traversals between the transmitter (i.e., the on-chip global buffers) and the receiver (i.e., the multipliers). Different kinds of network topologies have been proposed for them. For example, tree-based [11, 23], Benes-based [5, 7, 24, 29]

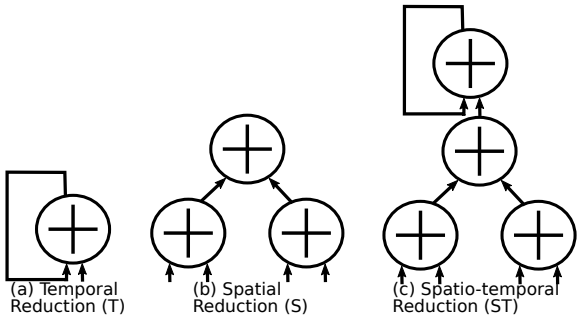

Fig. 2. Reuse patterns implemented in DL accelerators for partial-sum reduction.

or linear-based [9, 13, 18] topologies for the DN , enabling single-cycle traversals between the transmitter (i.e., the Global Buffer) and the receiver (i.e., the multipliers), 1D-linear [23, 29] or 2D-linear [9, 11, 18] topology for the MN, and customized tree-based topologies for the RN [23, 29]. The DN, MN and the RN can be independently reconfigured to efficiently map different dot product partitions through the creation of dynamic-size clusters.

The reduction stage through the RN is a critical part of the accelerator (it has a computation complexity higher than $O(1)$–see Figure 1) since it determines: (i) the flexibility of the architecture to map simultaneous variable-size cluster reductions (unfeasible in rigid accelerators); and (ii) the reuse pattern that will determine how the elements flow during the reduction, and therefore, the efficiency of the dot product reduction. These two features constitute what in this work we refer to as the *reduction dataflow*. To provide high energy-efficiency and low latency when processing the reduction dataflow, flexible DNN accelerators often dedicate a large part of the chip resources to implement the RN. For instance, MAERI [23], which clearly distinguishes between the 3 stages mentioned above, dedicates up to 25% and 38% of the chip area and power budget, respectively, just for this network.

Recent proposed accelerators implement one out of the three following reuse patterns which are directly linked to the way they manage folding situations: i) *Temporal (T) reduction*, in which each unit keeps the dot product to be reduced stationary in a register and performs a temporal accumulation of the different psums over that register (represented in Figure 2a); ii) *Spatial (S) reduction*, which coordinates a network of adder units, usually with a tree topology for efficiency, to perform a spatial reduction (illustrated in Figure 2b); and iii) *Spatio-Temporal (ST) reduction*, that combines both approaches usually performing a spatial reduction first followed by a final temporal one to accumulate different folding iterations (Figure 2c). Among the three approaches, the latter has been shown to reach better performance [21]. However, as we will demonstrate, adding Accumulators (Ac) to the RN for the T reduction as used in ST-based rigid accelerators, entails significant area and power when it comes to flexible accelerators.

In this work, we present a new strategy for Spatio-Temporal reduction in flexible DNN accelerator architectures that dodges the need of adding extra accumulators while keeping performance. In particular, our proposal, that we call STIFT (*Spatio-Temporal Integrated Folding Tree*), binds both spatial and temporal reductions together in a new design of the RN. STIFT builds on the observation that when several clusters are configured, there are some adder units in a tree-based RN topology that go unused. By adding a second root to the tree and additional links between the adder units, we create a new RN topology that is able to perform both spatial and temporal accumulations for all possible cluster configurations. This way, STIFT enables efficient and flexible dot product reductions while reducing the area and power dissipation up to 32% and 31%, respectively, with respect to the state-of-the-art (a RN augmented with accumulators).

Acronym	Reuse Pattern	Topology	Time	Flexibility	Hardware overhead
PT	T	N/A	$O(n)$	Rigid	Low
ST-Linear	ST	Linear	$O(n)$	Rigid	Low
S-Tree	S	Tree	$O[(i+l) \times \log_2(n/i)]$	Flexible	Low
ST-Tree$_{ac}$	ST	Tree	$O(i \times \log_2(n/i))$	Flexible	High
STIFT	ST	Tree	$O(i \times \log_2(n/i))$	Flexible	Low

Table 1. RNs implemented in DNN accelerators. *n=Elements in a dot product; i=Folding iterations.*

A preliminary version of this work has been accepted for publication in [28]. Here, we have extended that work with the following contributions:

- We describe in more detail our proposal and introduce new examples of mappings. These examples further clarify the way STIFT works and how it differs from previous implementations such as an ART.
- We extend the synthetic evaluation section by adding results for multiple same-size and multiple-size clusters, supporting our claims with results in terms of number of cycles per unit area. These new numbers demonstrate that STIFT is 3.30× and 1.48× more efficient than *S-Tree* and *ST-Tree$_{ac}$*, respectively, across different cluster configurations.

The rest of the paper is organized as follows. Background about contemporary strategies to implement reduction dataflows and their associated RN fabrics in both rigid and flexible accelerators are described in Section 2. We detail the architecture, operation with several mapping examples, and properties of STIFT in Section 3. Section 4 explains our evaluation methodology that involves RTL implementations and cycle-level microarchitectural simulation with STONNE. Our experimental results showing the benefits obtained in terms of performance, on-chip area and power consumption are exposed in Section 5. Finally, Section 6 summarizes our main conclusions.

2 BACKGROUND AND RELATED WORK

Reduction dataflows may be materialized in different RNs according to the flexibility of the accelerator design to create a variable number of clusters of arbitrary size to be reduced in parallel. In this section, we do a comprehensive literature review classifying how existing DNN accelerators give hardware support for reduction dataflows. Thus, we distinguish between the RNs in rigid accelerators, which limit the number and size of the clusters, and the RNs in flexible accelerators, which allow to create arbitrary number of clusters of varying size.

Along this section, we will utilize the example depicted in Figure 3a, wherein two groups of 8 psums (that have been previously generated by the MN to compute two different dot products) must be accumulated to obtain two output values. For each RN solution, we will use Table 1 to discuss different aspects such as the reuse pattern employed, the configured topology, the performance exhibited, its flexibility to map different number of dot products of arbitrary size, and the hardware overhead required to implement the folding support.

Reduction Networks in Rigid Accelerators. First-generation rigid DNN accelerators build on fixed-size clusters of MAC units interconnected by means of a fixed tightly-integrated on-chip network fabric. This means that the distinction between the MN and RN is purely logical, but for the sake of the simplicity, in this work we will focus on the accumulation process, which is the one that dictates the behaviour of the execution, and in the end, what defines its

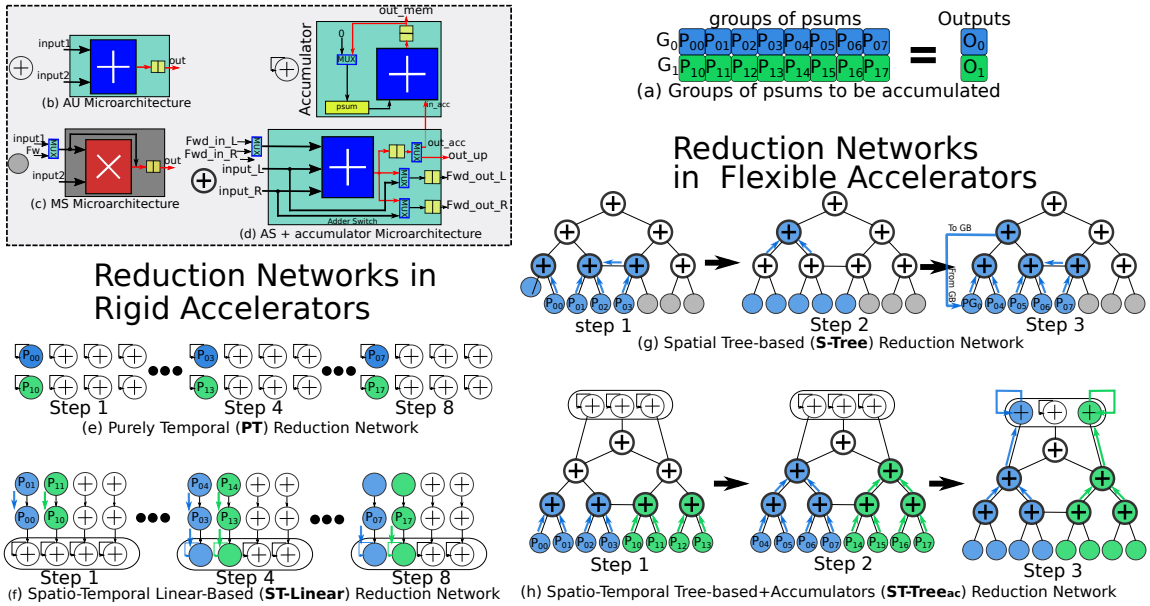

Fig. 3. Different types of reduction networks employed by state-of-the-art accelerators.

energy efficiency. Two RNs are typically employed in this type of accelerators: what we call a Purely Temporal (*PT*)[1] RN and a Spatio-Temporal Linear-based RN (*ST-Linear*)[2].

A PT RN is shaped by a set of non-connected accumulators which are Adder Units or AUs (see Figure 3b) augmented with a register for implementing a temporal reuse pattern in charge of temporarily reducing a particular dot product (see Accumulator in Figure 3d). This way, each accumulator has to fold $O(n)$ times to compute in $O(n)$ cycles a whole dot product that requires n multiplications, and the maximum number of dot products that may run in parallel is dictated by the number of accumulators in the RN. Figure 3e shows a 3-step walk-through example. As we may appreciate, the two dot products to be computed are mapped onto two accumulators. In step 1, the two units are initialized with the first psums. Then, the following steps accumulate the next psums, performing one sum per adder every cycle. Step 4 corresponds to the accumulation done in cycle 4, while step 8 finalizes the calculation after 8 cycles. This approach has been used by several prior accelerators such as [13, 26].

Contrarily, some accelerators such as [6, 18], exploit spatial reduction utilizing a Spatio-Temporal Linear-based RN (ST-Linear). This RN relies on several columns of simple AUs (see Figure 3b) connected vertically through a 1-D linear array (see Figure 3f). Each column of the topology corresponds to a cluster of AUs able to map and spatially reduce a single output vertically. Since it is very common to find that the number of AUs located in a column is smaller than the number of psums needed to be accumulated for a certain output (i.e., folding is needed), these networks rely on a Spatio-Temporal accumulation scheme to manage the folding accumulation process. With this aim, each column of AUs is extended with an extra accumulator that is in charge of performing temporal accumulation of the spatially accumulated iterations[3]. This way, similar to the PT RN, the total amount of time for a cluster of adders to calculate

[1]Observe that this corresponds to an Output Stationary (OS) dataflow in [31].
[2]This could link, for example, with the Weight Stationary (WS) dataflow in [31].
[3]Note: accelerators such as Eyeriss [9] do not have such accumulators for folding support and would be purely Spatial Linear.

a dot product is $O(n)$. Differently, the fact of employing a spatial reduction may increase unit utilization in several occasions. Figure 3f depicts an example. Here, the two outputs to be reduced are mapped onto the two first columns of the 2×4 topology and the reductions flow vertically during the three steps until the entire operation is reduced.

Both PT and ST-Linear RNs exhibit two main drawbacks: First, they are composed of a rigid interconnection network that makes them unable to gracefully adapt to different dot product dimensions and sizes. This has already been shown to lead to under-utilization of the computing units, which can significantly impact energy efficiency [10, 21, 23, 29]. The second drawback is that n-size reductions take $O(n)$ cycles to complete, which can be further improved by naturally performing the reductions in a tree-based manner

As it may be appreciated, both PT and ST-Linear RNs are composed of a rigid interconnection network that makes it difficult adaptation to different dot product dimensions. This leads, in general, to under-utilization of the units, which in turn translates into significant energy inefficiency of the architecture when running a particular DNN model [29]. To solve this problem, flexible architectures has recently emerged which incorporate configurable RNs able to accommodate a variable number of arbitrary-size clusters.

Reduction Networks in Flexible Accelerators. In order to overcome the limitations presented in rigid accelerators, recent *flexible* accelerators advocate having physically separated and reconfigurable DN, MN and RN fabrics. The RN is built from more configurable AUs called Adder Switches (ASs). Each AS is an AU augmented with a tiny switch to enable arbitrary cluster reductions of variable size over the same physical RN (see the AS block in Figure 3d), thereby significantly increasing unit utilization [23, 29].

The type of RN that materializes this flexibility is a Spatial Reduction Tree (S-Tree) used in both the ART and the FAN RNs proposed for MAERI [23] and SIGMA [29] accelerators, respectively. This type of RN enables efficient reduction by employing a binary tree-based accumulation so that an ideal whole reduction operation should take $O(\log_2 n)$ to be completed (see next for further details). Besides, to enable the parallel execution of any number of arbitrary-size clusters, they utilize augmented links (see the horizontal extra link between the two central ASs in the example of Figure 3g) which avoid conflicts when multiple reductions are mapped in the tree. The challenge to address, which is not analyzed with detail in any of these works, is how to manage the common situation of folding, in which the number of multiplications in a dot product is larger than the number of multiplier units implemented in hardware. In this specific purely spatial approach, which is the one implemented by these RNs [2, 4], the psum obtained in each cluster iteration is spatially sent to the Global Buffer (GB), and subsequently redistributed from it to a dedicated Multiplication Switch (MS) (see Figure 3c to observe the microarchitecture details of an MS), responsible only for forwarding it back to the RN. This way, the entire accumulation process is performed spatially. This is illustrated in Figure 3g. The leaves of the trees are MSs that just generated the psums. The values for the first iteration are initially sent and computed in steps 1 and 2. Then, for computing the second iteration, the value of the psum calculated in step 2 is passed through the GB in step 3, from where it is injected back into the RN by using a dedicated MS (the leftmost one) that just acts as a forwarder. The process is repeated until the whole dot product is spatially computed.

We observe in our evaluation (see Section 5) that this strategy, yet simple to implement in hardware, has two major drawbacks:

i. Low theoretical utilization of the MSs, as the required extra MS impedes to map efficiently some number of dot products. Note that, in our example, the extra MS needed to forward the psum impedes using the rest of the multipliers to calculate the other dot product, which also would need 5 multipliers.

ii. Low effective utilization of the mapped MSs as this implementation impedes to iterate over the same dot product in a *pipelined manner*. In the example, the MSs would not be able to operate in step 2, as the psum required for the

second iteration has not been calculated yet. In the general case, the time needed to compute an entire operation is $O[(i+l) \times \log_2(n/i)]$ as each of the iterations (i) will require to wait as many cycles as levels in the tree (l).

We would like to emphasize that these two drawbacks had not been observed yet in the previous works [23], and reveal the importance of proper folding management in flexible accelerators.

In order to overlap multiplications and sums of consecutive iterations of the same dot product, and thus, be able to attain a seamless pipelined execution for folding, it is necessary to break the dependency between two consecutive iterations by composing a Spatio-Temporal Tree (*ST-Tree*). To this end, one particular extension over the previous S-Tree, leveraging the design discussed for some ST-based rigid accelerators (Figure 3f), could be to add a set of accumulators, connected with the ASs, in charge of temporarily accumulating the different psums being calculated for each cluster. We call this approach an ST-Tree+Accumulators *(ST-Tree$_{ac}$)* RN. In this way, the time needed to complete an n-size operation in this case is $O(i \times \log_2(n/i))$. Besides, having these accumulators would eliminate the need to allocate the additional MS in each cluster just to forward the psum, thus making much better use of the MSs.

Figure 3d illustrates the microarchitectural details of an AS connected to its corresponding accumulator, able to perform temporal accumulation for a cluster that collapses on that particular AS.

As a representative example, Figure 3h shows how the accumulators would be employed with two configured clusters. In this case, there are 3 units, each of them connected to a different AS. As it may be appreciated, the first two steps are used to calculate the two psums of the first iteration (one per cluster). Observe that the calculation of the products required in the second iteration starts while the tree is computing the psums corresponding to the first one (step 2). Then, in the third step, these two psums are stored in the two accumulators, and at the same time, the tree of ASs computes the psums corresponding to the second iteration. In a fourth step (not shown in the figures for the sake of brevity), the result of the psums of the second iteration would be sent to the corresponding accumulators, where the final outputs would be obtained.

This approach has two major inconveniences. First, and most importantly, it entails significant area and power overhead as it requires to significantly increase the number of adders in the RN (i.e., the extra accumulators). The second inconvenient has to do with the resulting increased complexity of the design by requiring the addition of this extra component.

3 STIFT: A SPATIO-TEMPORAL INTEGRATED FOLDING TREE

The discussion presented in the previous section reveals that Spatio-Temporal RNs are the best match for both rigid and flexible DNN accelerators by endowing the RN with accumulators. To avoid incurring into high on-chip area and energy overhead due to these extra components, the number of accumulators and the connections to the ASs should be done according to the size and number of the clusters that are supported by the architectural design.

In the case of rigid accelerators, for example, in an R×C ST-Linear RN like the one used in the Google's TPU, the number of the accumulators should be equal to the number of columns (i.e., C) of the design (see Figure 3f), as a whole column is typically used to vertically reduce a cluster. This way, in this type of architectures the accumulation units are affordable and can be implemented without incurring significant area and energy overhead.

On the other hand, in the case of the higher-performance flexible accelerators, since the number, size and exact location of the clusters to be mapped onto the architecture is undetermined, it is not known *a priori* the number of accumulators that are needed to support all the configurations. Therefore, in order to support the worst case in which every multiplier was an independent cluster, the architecture would need as many accumulators as ASs, doubling

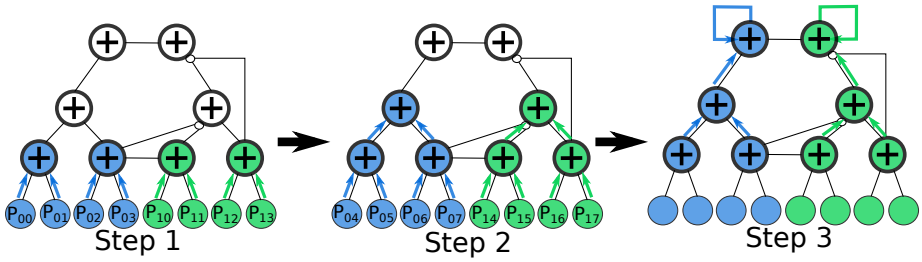

Fig. 4. STIFT running the example shown in Figure 3a.

the area required for the RN. Reducing the number of accumulators could lower this overhead, but it would limit the number of clusters that can be accumulated in parallel, constraining the flexibility.

To ensure efficient folding support in flexible accelerator architectures and to avoid the addition of such extra accumulators, we propose in this work a novel Reduction Network fabric, specifically suited for flexible accelerator architectures. We call our proposal STIFT which stands for *Spatio-Temporal Integrated Folding Tree*. Similarly to ST-Tree$_{ac}$ RN using accumulators, STIFT is capable of running any number of dynamic-size clusters in a non-blocking manner, but unlike this one, it enables efficient and flexible support to ensure full non-blocking processing of folding.

3.1 STIFT Topology

The observation behind STIFT is that for the S-Tree RN employed in recent accelerator proposals [23, 29], there are free ASs when two or more clusters are configured (for the example in Figure 3h, the AS at the root of the tree is not used as two clusters have been defined). The more (and in consequence smaller) clusters that are formed, the more ASs go unused. So, the immediate question is: *why not dedicate these free ASs to accumulating the psums for each cluster?*

To do so, we obviously need that the RN can guarantee, at least, one free AS per each cluster. Taking a look at Figure 3h, we can clearly see that the number of ASs in the tree is insufficient for our purpose (note that when the two clusters are formed, just one AS becomes free). We can also see that we can easily solve the problem by adding a second root to the tree, and also several links to guarantee that every cluster, independent of its configuration (size and/or location), can always have access to one of the free ASs that will be used to accumulate the corresponding psums. In broad terms, this is STIFT.

Before delving into the details of how the extra links required by STIFT are added, Figure 4 shows an example of how folding would be carried out with STIFT. In this case, the ASs have to be extended in order to be able to act either as accumulators or ASs (further details in Section 3.2). We call these units Extended Adder Switches (eASs). As we can see, the top eASs are employed to accumulate the two dot products being calculated. This way, in this case, after step 2, the eAS on top of each cluster sends its psum to the corresponding root of the tree, which will be configured to accumulate the psums calculated in the two iterations. Once the calculation of the dot products is finished (all the psums have been accumulated in each case), these nodes will send the accumulated values to the Global Buffer, and then, they will begin to accumulate the psums of the next dot product assigned to each of the clusters. Note that, as with ST-Tree$_{ac}$, STIFT removes the dependency among consecutive iterations, and hence, enables fully pipelined execution in the MSs and eASs. However, unlike ST-Tree$_{ac}$, STIFT does not require the addition of extra accumulation units, but instead achieves the same goal by only requiring some lightweight extra logic to leverage the ASs to play the role of both ASs and accumulators, one extra root node, and some extra connections between the eASs.

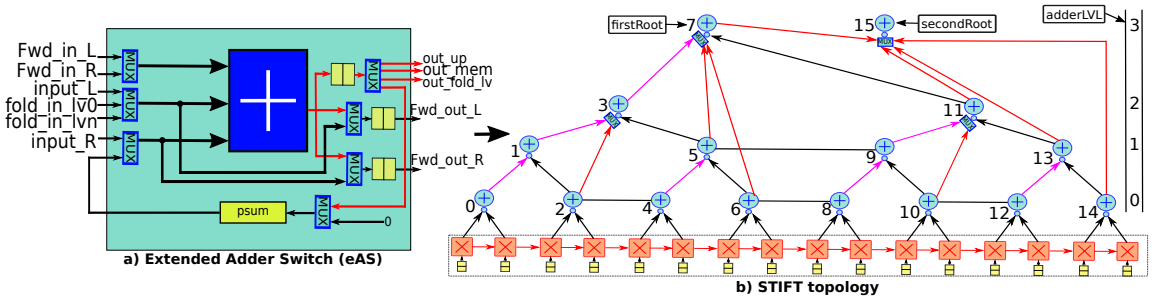

Fig. 5. a) Extended adder switch microarchitecture of STIFT. b) STIFT topology. The black and pink links are shared with ART, but the pink ones are also used for folding purpose. The red links are additional folding links introduced in STIFT to remove all structural hazards during the folding process.

A detailed view of a 16-wide STIFT topology is depicted in Figure 5b. All the links that are inherited from the S-Tree RN topology that we use as base (the ART network [23]) for building STIFT are shown either in pink or black colors. In red, we identify the new links that STIFT requires and that we call the *folding links*. We add the minimal number of links to ensure that every cluster can use an eAS to accumulate its psums and produce its final output value. This is achieved by using either a pink or a red link.

In Algorithm 1, we show how to add the extra links required by STIFT (the red links in Figure 5b) for an arbitrary number of adders (`numAdders`). According to this algorithm, for each of the nodes in the topology (i.e., *i*), we iterate over the levels (i.e., *lvl*) that are below the node to connect its corresponding pair in that level. By establishing the new links in this way, we guarantee that every eAS in the RN can have a different associated parent eAS that would be used to perform the accumulation process for the cluster that converge in the first eAS.

3.2 Microarchitectural implications of STIFT

Figure 5a shows the microarchitectural details of each of the eASs used in STIFT. As observed, it is quite similar to the ASs of a flexible RN such as ST-Tree$_{ac}$ (and that we explained in Section 2). Since the proposed STIFT topology enables integrated temporal accumulation, every of its eAS has an extra register which is used to store the accumulated partial sum in case it is needed (the eAS acts as the accumulation point for a particular cluster). Besides, since the left input data (see Figure 5b) could be injected either from the left-child eAS or from any other lower-level eAS which sends the psum to be accumulated, it is necessary to incorporate an extra multiplexer unit selecting this input. The size of this multiplexer varies according to the level in which the eAS is located in the reduction tree. In general, given an eAS at a

```
//Code to add STIFT links
//Inputs: numAdders, adderLVL
for(int i=0; i < numAdders-1; i++):
 for(int lvl=1;lvl<adderLVL[i];lvl++):
  connect adder i with: i - 2^(lvl-1)
//Connecting first with second root
int firstRootID=numAdders/2 - 1
int secondRootID=numAdders / 2
Connect firstRootID with: secondRootID
```

Algorithm 1. Pseudocode illustrating how the additional links required by STIFT are added.

certain level L, the size of its multiplexer will be *L-1:1*, as every level in the tree will have a certain eAS connected that will provide the psums to accumulate (when the eAS is used for this purpose). In order to avoid the critical path delay degradation due to the long wires connecting the different levels in the topology, all of the inter-eAS/MUX wires are implemented in the semi-global metal layers using standard repeater wires.

Each eAS in STIFT can act either as an AS (psum spatial generator) or as an accumulator (psum temporal reducer). If the eAS is configured as an AS, then it will receive the inputs from the children eASs, will calculate the psum, and will send the output to its parent eAS (as done in S-Tree). On the contrary, if the eAS is configured as an accumulator, the left input will be obtained from the eAS that sends the psums, and the right input will be taken from the internal register. Once the final accumulation has been performed, the result will be sent directly to the Global Buffer. All the additional multiplexers, as well as the new operation mode are configured by the mapper (see Figure 1) based on the size and number of the clusters to be reduced. The benefit of this approach is that since STIFT topology allows the eASs to operate either in psum spatial generator or psum temporal reducer modes, they can reuse the adder unit already contained in the eAS, reducing significantly the area overhead and power dissipation in comparison to adding the extra accumulators required in the ST-Tree$_{ac}$ RN (see Section 5).

3.3 STIFT Mappings

3.3.1 Examples of diverse mapping configurations and reduction dataflows with STIFT. Figure 6 depicts 5 possible mapping configurations in a 16-wide STIFT RN to perform the accumulation of 8 16-size groups of psums (that have been previously generated by the MN to compute 8 different dot products). As we can see, the flexibility of STIFT allows not only to map arbitrary size clusters, but also to perform temporal accumulation for all of them. Next, we present in detail each mapping configuration. For ease of explanation, we have labeled each node in the tree with a different number.

Mapping 1: This case is the simplest possible mapping configuration as an entire group of psums is reduced by mapping it onto all the available MSs in the MN fabric (16). As we can observe, the cluster collapses in the first root of the tree (i.e., node 7) and the psums are forwarded to the second root (i.e., node 15) to be reduced in a pipelined manner. Since in this case the cluster size equals the group size, there is just one iteration (i.e., temporal accumulation is not required), and the only psum generated will be directly forwarded to main memory. Subsequent groups of psums will be mapped to the MSs similarly, generating, one after other, the eight final reductions.

Mapping 2: In this particular example, two groups of psums are mapped for reduction onto STIFT. As mapping the entire two groups would require 32 MSs, this reduction has to be divided into two 8-size clusters. This means that each cluster requires 2 iterations which are temporarily accumulated by the two roots of the tree (i.e., nodes 7 and 15), which are configured as accumulators. In this figure, we can note the way in which eAS labeled with 11 requires the use of the folding link to forward the generated psum to node 15. Sending the psum to any other node would produce a structural hazard (the corresponding adder in the eAS has to carry out two different psums) which would break the computation pipeline.

Mapping 3: In this mapping, four groups of psums are allocated to the 16 MSs. To do so, the computation is divided into four 4-size clusters, each performing 4 iterations in total. As it may be appreciated, the 4 clusters collapse in the 4 eASs belonging to the first level of the tree, and therefore the use of folding links by the nodes 5 and 13 is required to avoid the kind of structural hazards in the upper levels already illustrated in the previous mapping.

Mapping 4: This is the extreme case of mapping in which eight 2-size clusters are configured to reduce 8 groups of psums, each requiring 8 iterations. Even in this case in which the 8 clusters are spatially collapsing in the 8 eASs

belonging to the first level of the tree, there are always free wires and eASs to send the corresponding psums and carry out the temporal accumulations, respectively, in a pipelined manner.

Mapping 5: This case illustrates how STIFT can be utilized to map arbitrary size clusters. This enables to efficiently support scenarios in which sparsity is exploited. In these cases, the matrices are compressed (e.g., in CSR or bitmap formats) to reduce computation and memory footprint, and the resulting dot products can have variable size. In this example, four groups of psums are mapped to be reduced, but they present different sizes. As we can see, the groups 0, 1, 2, and 3, with cluster sizes of 3, 5, 2 and 6, respectively, are mapped, which require 5, 3, 7 and 2 iterations, respectively. Like the ART, STIFT leverages the intermediate links between eASs that do not share the same parent for mapping irregular cluster sizes. Even in this case, STIFT supports temporal accumulation by taking advantage of the novel folding links.

3.3.2 Generating mapping signals. Given a particular mapping, the control logic signals to be sent to STIFT RN to route the psums can be generated either offline or at runtime by following the steps given as follows. First, the ART routing decisions are made by applying the particular ART algorithm (described in [23]). This algorithm ensures that the clusters are able to reduce in a non-blocking manner for a single iteration. Once this process is completed, the eASs where the clusters are collapsing are configured. If the position of a particular eAS in the level of the tree is even, the psum is configured to be sent to the parent using the ART link. Otherwise, the eAS is configured to forward the psums to the folding link.

4 EXPERIMENTAL METHODOLOGY

To prove the benefits of STIFT as the RN of a flexible accelerator architecture, our evaluation methodology considers the following three different angles: 1) RTL implementation to faithfully obtain both energy and area numbers; 2) synthetic evaluation to describe the main benefits of STIFT; and 3) end-to-end evaluation to demonstrate how STIFT can be used to run real DNN models. Below, we describe each of these three angles in detail.

4.1 RTL Implementation

We analyze the power and on-chip area overheads that the different RN solutions entail. In particular, we have implemented both STIFT[4] and ST-Tree$_{ac}$ RNs in BSV (Bluespec System Verilog) [1] using INT16 datatype, and use Synopsys Design Compiler and Cadence Innovus Implementation System for synthesis and place-and-route, respectively, using TSMC 28nm GP standard LVT library at 800 MHz. For comparison, we also consider MAERI's ART design [2] as an example of S-Tree RN. We study how the different RN fabrics scale by exploring different RN widths (number of multiplier units) and data formats (i.e., INT16, FP16 and FP32). To explore different data formats, we have run synthesis and place-and-route for both FP16 and FP32 ALU units and analytically added the area and energy overhead to our INT16 designs.

4.2 Synthetic evaluation

We conduct a comparison of the performance (runtime) that the three flavours of RNs (S-Tree, ST-Tree$_{ac}$ and STIFT) achieve for reduction operations. To do so, we have implemented these three RNs in STONNE simulator [27]. STONNE is a cycle-level, highly-modular and highly-extensible microarchitectural simulation framework that can be potentially plugged into any high-level DL framework as an accelerator device. Therefore, all the activity found during the

[4]STIFT is available on https://github.com/maeri-project/stift-bsv.

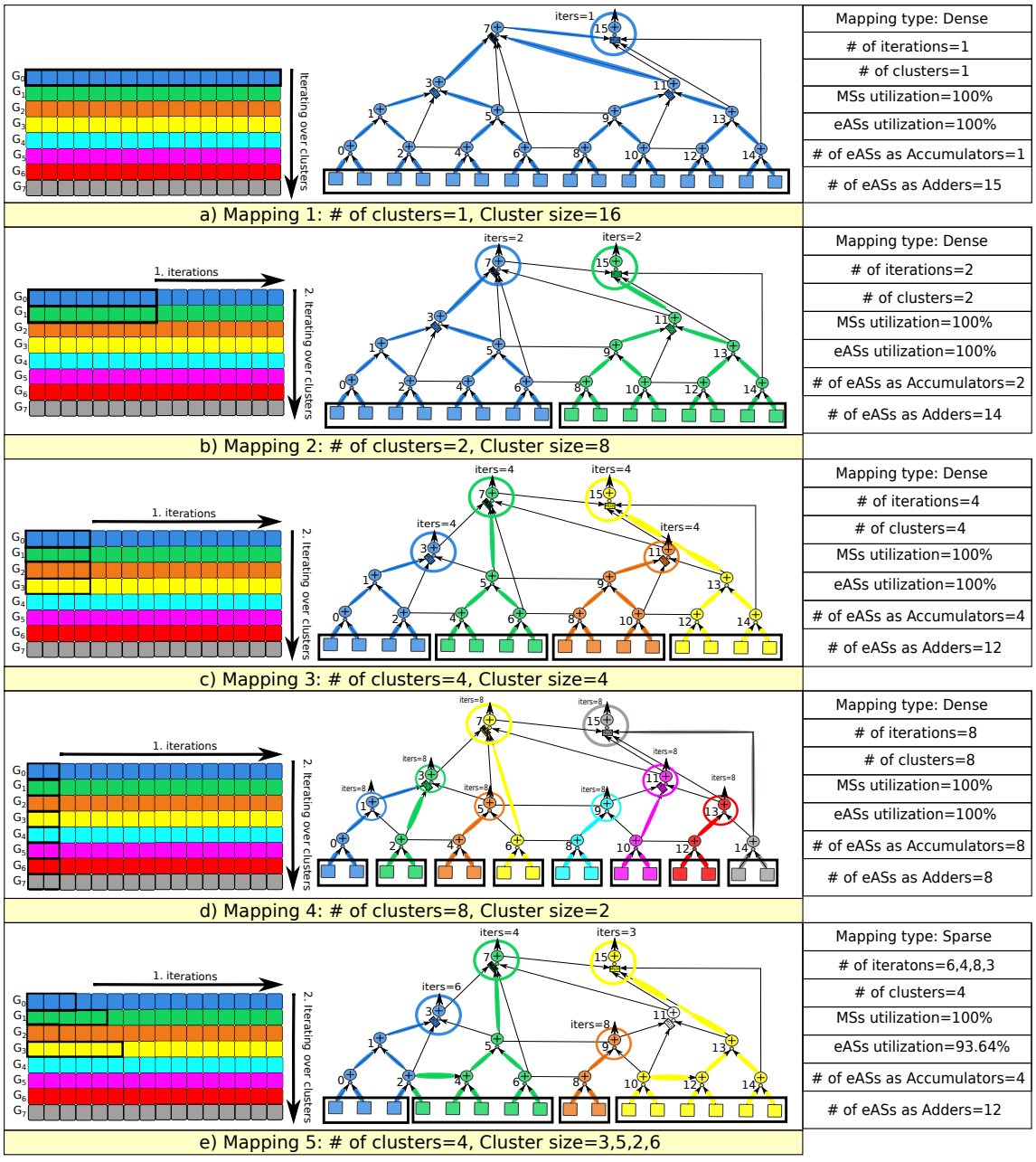

Fig. 6. Different mapping configurations to perform 8 16-size cluster accumulations on a 16-wide STIFT.

execution of the DNN inference procedure of real, unmodified DNN models processing real input data can be accurately simulated at cycle level. STONNE also allows us to easily configure and implement different reduction networks, thereby facilitating the task of evaluating and comparing our three target RNs.

For a fair comparison, we have configured the simulator with the same DN and MN networks in all cases. In particular, we consider a tree-based DN and a linear-based MN to build MAERI-like flexible accelerator architectures [23]. Besides, we assume the next hardware parameters: 256 MSs, FP16 arithmetic datatype, 108-KiB Global Buffer size, 128 elements/cycle I/O Global Buffer bandwidth, and two 256 GB/s, 512-MiB HBM2.0 DRAM modules. For the resulting three accelerator architectures, we run three different experiments:

1. First, to study how a single cluster behaves, we have executed seven synthetic workloads by mapping a single cluster ranging in size from 2 to 128. To study how each RN deals with folding, we iterate each cluster 512 times [5], which allows us to observe how the RN topology affects performance when running a certain representative reduction.

2. To observe the impact of mapping multiple same-size clusters in parallel when it comes to folding, we have executed other seven synthetic workloads. Differently, this time we have swept the number of clusters to be mapped and its size, but keeping the number of used multipliers as 128 in all the cases. This way, we have executed the next configurations: 64 clusters of size 2 (*64C-S2*), 32 clusters of size 4 (*32C-S4*), 16 clusters of size 8 (*16C-S8*), 8 clusters of size 16 (*8C-S16*), 4 clusters of size 32 (*4C-S32*), 2 clusters of size 64 (*2C-64S*) and 1 cluster of size 128 (*1C-128S*). Note that we always use the same terminology *xx*C-*yy*S to refer to *xx* mapped clusters and its size (*yy*). Similar to the aforementioned experiment, we iterate each cluster 512 times.

3. Finally, we demonstrate that STIFT is able to map multiple variable-size clusters in parallel and still perform the folding efficiently. In this way, in this experiment we configure irregular-size clusters keeping always the utilization of 128 multipliers and iterate each cluster 512 times.

4.3 End-to-end evaluation

We have determined the overall impact on performance and energy consumption of the different RNs when running complete DNN workloads. To do so, we have used STONNE together with its Pytorch interface [3] to execute the inference procedure of seven full DNN models on the three resulting accelerator configurations (S-Tree, ST-Tree$_{ac}$ and STIFT). We consider *Alexnet* [20] (*A*), *Mobilenets* [16] (*M*), *Squeezenet* [17] (*S*), *Resnet-50* [15] (*R*), *VGG16* [30] (*V*), *ssd-Mobilenets* [25] (*S-M*) and *BERT* [12] (*B*), taken from MLPerf [14] benchmark suite. In all the cases, we have obtained the best mapping configurations using the mRNA tool [32]. To calculate the energy consumed by each execution, we have fed the simulator with the energy numbers obtained during the RTL synthesis and presented in Section 5.

5 RESULTS

5.1 RTL evaluation

Figure 7 shows the area (μm^2) and power (mW) synthesis results for the three evaluated RN designs: *S-Tree*, *ST-Tree$_{ac}$* and *STIFT*. For both area (top) and power (bottom) figures, we plot three different design points varying the data type that the RN utilizes (INT16, FP16 and FP32). For each one of them, we also show in the *x-axis* the results for 5 RN sizes by scaling the number of MSs (i.e., the leaves) from 64 to 1024. Each bar is split into two components: the amount of area/power attributable to the Adder Units (*AUs*) and that corresponding to the rest of elements of the ASs and the topology, namely switching logic, registers, multiplexers and wires (*Rest*).

As it may be appreciated, for INT16 (see Figures 7a and 7d), which is a datatype typically used in some inference accelerators (e.g. TPUv1 [18] can operate on 8- and 16-bit data elements), we observe that STIFT saves up to 19% and 20% power and on-chip area, respectively, as compared with *ST-Tree$_{ac}$*. In this case, the simple INT16 AUs do not

[5] We observe this is a representative folding iteration number when running real DNN workloads.

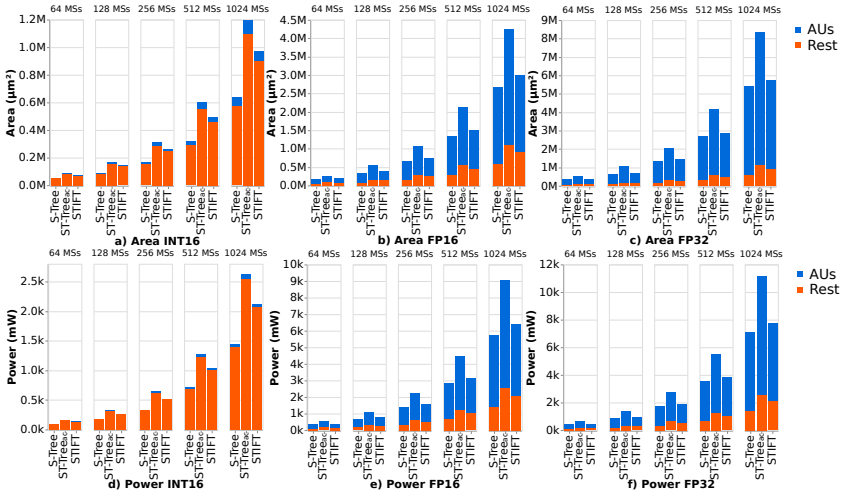

Fig. 7. Area (μm^2) and power (mW) synthesis results for the three evaluated RN designs (*S-Tree*, *ST-Tree*$_{ac}$ and *STIFT*) varying RN size (64, 128, 256, 512 and 2014 MSs) and data types (INT16, FP16 and FP32).

MSs	AUs S-Tree	AUs ST-Tree$_{ac}$	AUs STIFT	Wires S-Tree	Wires ST-Tree$_{ac}$	Wires STIFT	Muxes STIFT
64	63	126 (+100%)	64 (+2%)	152	215 (+41%)	184 (+21%)	63
128	127	254 (+100%)	128 (+1%)	311	438 (+40%)	375 (+20%)	127
256	255	510 (+100%)	256 (+1%)	630	885 (+40%)	758 (+20%)	255
512	511	1022 (+100%)	512 (+0.1%)	1269	1780 (+40%)	1525 (+20%)	511
1024	1023	2046 (+100%)	1024 (+0.1%)	2548	3571 (+40%)	3060 (+20%)	1023

Table 2. Number of MSs, AUs, wires and multiplexers (Muxes) required to implement *S-Tree*, *ST-Tree*$_{ac}$ and STIFT RNs for different RN sizes. The percentage of extra elements added in *ST-Tree*$_{ac}$ and in STIFT with respect to *S-Tree* is also shown.

represent a significant fraction of the logic of the RNs (see blue portion of the bars), and therefore, the elimination of the extra AUs of the accumulators with STIFT brings modest savings. Reductions in power and on-chip area in this case come mostly from the elimination of the extra wires and logic needed to connect the accumulators when these units are used. The reduction in the number of elements can also be observed in Table 2, where we show the number of wires presented in each RN and the overhead added by *ST-Tree*$_{ac}$ and STIFT with respect to *S-Tree*. As we can see, STIFT not only reduces the number of AUs, but also the wires (average reduction of 20%), making it more efficient even when the area and energy fraction represented by the AUs is not very significant. This also demonstrates that our STIFT design will introduce less overhead as technology shrinks. In these cases, the energy spent in the wires becomes dominant and STIFT ensures high flexibility while keeping the RN more efficient. Finally, we can also observe that the overhead introduced by the muxes in the STIFT design does not translate into a scalability issue as the orange bar sizes increase linearly with the RN size for both area and energy numbers.

Power and area reductions are more significant when it comes to the FP16 and FP32 designs, as the AUs are responsible for the most important fraction of the chip area attributable to the RN. For example, for FP16 (mostly used in some inference devices, such as Eyeriss [9], for better inference accuracy) the AUs consume between 70% and 78% of the total chip area devoted to the RN. This results in STIFT obtaining significant area (Figure 7b) and energy (Figure 7e)

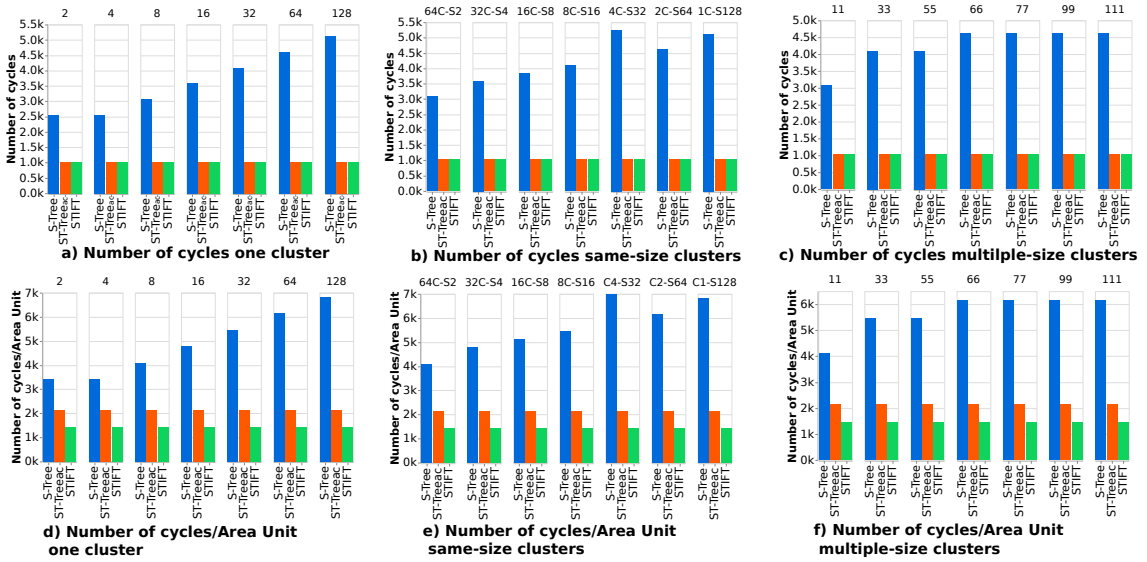

Fig. 8. a) Number of cycles for the three RNs (*S-Tree*, *ST-Tree*$_{ac}$ and STIFT) to complete the reduction of one single cluster varying its size. b) Number of cycles for the three RNs (*S-Tree*, *ST-Tree*$_{ac}$ and STIFT) to complete the reduction of multiple same-size clusters varying both the number of clusters (C) and its sizes (S). c) Number of cycles for the three RNs (*S-Tree*, *ST-Tree*$_{ac}$ and STIFT) to complete the reduction of multiple variable-size clusters varying both the number of clusters and its sizes. d-f) Number of cycles per unit area for the experiments performed in *a-c*.

reductions over ST-Tree$_{ac}$, which range from 28% to 30% and from 29% to 30%, respectively. As shown in Figures 7c and 7f, these differences are even larger for FP32 (utilized in training accelerators). In this case, 87% on average of the chip area occupied by the RN is due to the AUs, and STIFT reduces area and energy demands of ST-Tree$_{ac}$ by 32% and 31%, respectively, on average.

Obviously, when compared to S-Tree, STIFT introduces area and power overheads, as it adds the accumulation logic needed to perform temporal reductions. On average across all the design points, the extra power and area overheads are 17% in both cases, much lower than the 39% and 40%, respectively, added by the accumulation buffer in ST-Tree$_{ac}$.

Nevertheless, as we next show, unlike S-Tree and similar to ST-Tree$_{ac}$, STIFT enables pipeline execution of consecutive folding iterations, resulting in much better performance results than S-Tree (and, also, less amount of total energy consumed).

5.2 Synthetic evaluation

Figure 8a shows the number of cycles (*y-axis*) obtained for the three RN implementations to complete the reduction of a single cluster varying its size (as described in Section 4.2). Across the seven considered sizes, we observe that both ST-Tree$_{ac}$ and STIFT are on average 3.43× faster than S-Tree, demonstrating that when the reductions are performed purely spatially (and the psums are forwarded through the Global Buffer for supporting folding), performance degrades significantly due to the psum dependency between consecutive folding iterations.

The performance gap between ST-Tree$_{ac}$/STIFT and S-Tree is more significant as the cluster size increases. For the smallest cluster size (i.e., 2), we appreciate that ST-Tree$_{ac}$ and STIFT are 2.49× faster than S-Tree, while this difference

reaches 4.95× for a cluster size of 128. The reason for this is that the higher the reduction tree, the longer it takes to produce the psum and return it to the MN *via* the Global Buffer, and therefore the longer the MN will have to wait until it can start the next folding iteration. Another important limitation to note is that when folding is used, S-Tree must dedicate one MS per cluster to forward the psums calculated in the previous iteration. This results in worse utilization of the MSs, which usually translates into a larger number of folding iterations required per cluster (due to smaller effective cluster sizes). On the contrary, spatio-temporal approaches like ST-Tree$_{ac}$ and STIFT eliminate this constraint as the accumulation of consecutive folding iterations is performed temporarily on the top of each tree, leaving the entire MN available for effectual computation, and thus allowing for more flexible cluster reduction mappings.

On the other hand, Figure 8 shows the results for the second experiment described in Section 4.2 in which we sweep the number of mapped clusters and its size (*x-axis*). In this case, we keep all the clusters with the same size. As we can observe, similar to the first experiment, there is a clear gap between the number of cycles obtained with S-Tree and the number of cycles obtained with both ST-Tree$_{ac}$ and STIFT, verifying that STIFT is suitable to run multiple clusters in parallel. However, in this case the results are even more impressive with both STIFT and ST-Tree$_{ac}$ being 4.02× faster than S-Tree. This difference is due to both spatio-temporal approaches keep the partial sums stationary and therefore reduce the number of writings to memory when multiple clusters are reducing in parallel. This alleviates the pressure on the memory hierarchy which translates even to further performance and improvements.

The results for the third experiment described in Section 4.2 are shown in Figure 8c. Here, we vary again the number of clusters and its size, but this time we map variable size of clusters in parallel. The *x-axis* in this case shows the largest cluster configured in each execution. This is so, because we have clearly observed that when variable-size clusters are running in parallel, the largest one dominates the execution time while the rest lag far behind it. In this experiment we have observed that both STIFT and ST-Tree$_{ac}$ are 4.07× faster than S-Tree. Similar to the second experiment, mapping variable-size clusters when STIFT is configured benefit from alleviating the pressure on the memory hierarchy with respect to S-Tree. Furthermore, this effect is even exacerbated as the clusters write at different times due to its irregular size.

Finally, it also may be appreciated that STIFT and ST-Tree$_{ac}$ reach virtually the same performance numbers across the three experiments. This comes as no surprise if we take into consideration that both approaches implement spatio-temporal reductions and behave similarly. The difference in this case is that by integrating temporal accumulations in the RN, STIFT can save the significant area and power overhead that the use of the accumulators add to ST-Tree$_{ac}$. This is further demonstrated in Figures 8d-f where we show the number of cycles per unit area obtained when running the three aforementioned experiments depicted in Figures 8a-c, respectively. In this case, we have divided each number of cycles shown in the Figures 8a-c by the inverse of each RN area occupation given the results described in Section 5.1. We observe that on average across the executions of the three experiments, STIFT improves the number of cycles per unit area by 3.30× and 1.48× against S-Tree and ST-Tree$_{ac}$, respectively. Further experiments in next section with real DNN models validate this claim.

5.3 End-to-end evaluation

As for the performance observed for the three configurations under study with complete DNN models, Figure 9a shows the speed-ups obtained by ST-Tree$_{ac}$ and STIFT with respect to S-Tree. Our results corroborate the trend observed with the synthetic workloads in that we clearly see that the folding strategy of redistributing the psums through the Global Buffer and the extra MS, results in very poor performance, mainly due to the dependency created between consecutive iterations.

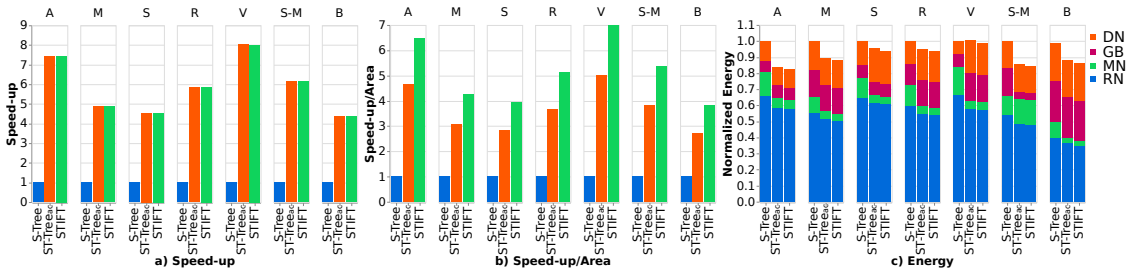

Fig. 9. a) Speed-up of ST-Tree and STIFT against S-Tree after running 7 DNN models. b) Speed-up/Area of ST-Tree and STIFT against S-Tree after running 7 DNN models. c) Normalized Energy of ST-Tree and STIFT against S-Tree after running 7 DNN models. *A=Alexnet; M=Mobilenets; S=Squeezenet; R=Resnet-50; V=VGG16; S-M=ssd-Mobilenets; B=Bert.*

By augmenting the S-Tree configuration with the accumulators, and therefore, implementing spatio-temporal reductions (ST-Tree$_{ac}$), performance speed-up of up to 8× for VGG-16 (5.9× on average across the 7 DNN models) are gained. It is also interesting to appreciate how the magnitude of the gains changes from one DNN model to another. For instance, for the largest networks, namely Alexnet, Resnet-50, VGG-16 and ssd-Mobilenets, the obtained speed-ups range from 5.86× to 8.03×. On the contrary, for the smallest networks, namely Mobilenets-V1 and Squeezenet the speed-up values are 4.88× and 4.54×, respectively, as their layers are smaller, and therefore, some of them do not make use of folding. In particular, Mobilenets-V1 and Squeezenet require folding in 75% and 39% of their layers, respectively, while this percentage grows to 99% for Alexnet, Resnet-50 and VGG-16. The only exception is BERT, which despite being the deepest network, its runtime is dominated by layers of size 768 (20% of the layers) which barely require 3 folding iterations, and by attention layers, which represent 75% of the total number of layers and do not require folding as their size is small (64). The achieved speed-up in this case is 4.37×.

As we explained in Section 2, the reason for these significant performance improvements is that the use of the accumulators allows to break the aforementioned dependency between consecutive folding iterations, thus enabling that the different components of the flexible ST-Tree$_{ac}$ architecture can operate in a pipelined manner (the DN, MN, RN and accumulators). But, what is more important, we can observe that the more area- and power-efficient STIFT design very closely approaches the speed-up values of ST-Tree$_{ac}$. In Figure 9b, we consider both achieved speed-ups and area requirements of each design. The area requirements are normalized with respect to the S-Tree case, which is also the reference for the calculation of the speed-ups. Here, we can clearly see that in all cases, STIFT reaches the best compromise between performance and area consumption (the higher Speed-up/Area values). This is due to, compared with ST-Tree$_{ac}$, STIFT achieves virtually the same performance but it requires significantly less area (see Figure 7b). Overall, we can appreciate that on average across the seven DNN models, STIFT reaches a speed-up/area ratio of 5.13× while this value is reduced to 3.67× in the case of ST-Tree$_{ac}$.

Finally, Figure 9c plots a breakdown of the total amount of energy consumed (static and dynamic energy consumption) in each case, distinguishing between the main components of the architecture. All the results have been normalized with respect to the obtained for S-Tree. As we can see, the energy benefits are not as impressive as the performance numbers. The reason for this is that energy consumption in each case is mainly dominated by the dynamic component (dynamic energy is 30× higher on average), and the number of operations that must be carried out is the same regardless of the implemented folding strategy. Nevertheless, the fact that the total number of execution cycles is greatly reduced

in STIFT and ST-Tree$_{ac}$ translates into significant reductions in the static component, which in turn results in average reductions of 11% and 9% in total energy (up to 20%) for STIFT and ST-Tree$_{ac}$ RNs, respectively.

6 CONCLUSIONS

In this paper, we have demonstrated that the way the Reduction Network (RN) of a flexible DNN accelerator architecture manages folding significantly impacts the achievable performance, when processing the inference procedure of DNN models. More specifically, a purely temporal approach is not able to perform the reductions in a tree-based manner, which limits its efficiency. On the contrary, a spatial reduction typically lacks of sufficient resources to reduce large clusters.

To overcome this, we have presented a new strategy for Spatio-Temporal reduction–STIFT–that dodges the need of adding the extra accumulators used by contemporary RN fabrics in flexible DNN accelerators. STIFT takes advantage of the unused adder units that are available in a tree-based RN topology when several clusters are configured, and adds the minimal elements required to guarantee that for all possible cluster configurations, each one can be associated with one adder unit for temporal accumulation of its generated psums.

Through a comprehensive evaluation that includes RTL implementation of different RN approaches able to operate with several data types, we have observed that STIFT obtains area and power benefits of up to 32% and 31% respectively. Overall, in the context of a specific flexible accelerator like MAERI, this translates into general area and energy savings of up to 8% and 12, respectively%[6]. Besides, through cycle-level simulation of complete and synthetic DNN models, we have proven that STIFT is up to 3.67× more efficient in terms of the balance between the performance and area requirements for the considered alternatives.

ACKNOWLEDGMENTS

We would like to thank Jianming Tong for his help during the RTL synthesis process and Hyoukjun Kwon for his feedback. This work was supported by grant RTI2018-098156-B-C53 funded by MCIN/AEI/10.13039/501100011033 and by "ERDF A way of making Europe", NSF OAC 1909900 and US Department of Energy ARIAA co-design center. Also, it was supported by project grant PID2020-112827GB-I00 funded by MCIN/AEI/ 10.13039/501100011033. F. Muñoz-Martínez was supported by grant 20749/FPI/18 from Fundación Séneca.

REFERENCES

[1] [n.d.]. Bluespec System Verilog (BSV). http://wiki.bluespec.com/.

[2] [n.d.]. MAERI code v1. https://github.com/hyoukjun/MAERI.

[3] [n.d.]. PyTorch. https://pytorch.org/.

[4] [n.d.]. SIGMA code v1. https://github.com/georgia-tech-synergy-lab/SIGMA.

[5] S. Arora, T. Leighton, and B. Maggs. 1990. On-line algorithms for path selection in a nonblocking network. *In Proc. of the 22nd annual ACM symposium on Theory of Computing* (April 1990), 149–158.

[6] Lukas Cavigelli and Luca Benini. 2016. Origami: A 803-GOp/s/W Convolutional Network Accelerator. *IEEE Transactions on Circuits and Systems for Video Technology* (July 2016), 2461–2475.

[7] A. Chakrabarty, M. Collier, and S. Mukhopadhyay. 2009. Matrix-based Nonblocking Routing Algorithm for Bene˘s Networks. *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns* (April 2009).

[8] Kun-Chih (Jimmy) Chen, Masoumeh Ebrahimi, Ting-Yi Wang, and Yuch-Chi Yang. 2019. NoC-based DNN accelerator: a future design paradigm. In *2019 IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. 1–8.

[9] Yu-Hsin Chen, Joel S. Emer, Tushar Krishna, and Vivienne Sze. 2017. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. *IEEE Journal of Solid-State Circuits* 52, 1 (Jan. 2017), 127–138.

[6]Note that MAERI is an accelerator targeting inference which implements the INT16 datatype. Other accelerators targeting training and using a wider datatype may experience larger benefits.

[10] Yu-Hsin Chen, Joel S. Emer, and Vivienne Sze. 2017. Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators. *IEEE Micro* 37, 3 (June 2017), 12–21.

[11] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 2019. Eyeriss v2: A Flexible Accelerator for Emerging Deep Neural Networks on Mobile Devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 9, 2 (June 2019), 292 – 308.

[12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv: 1810.04805v2 (2019)* (May 2019).

[13] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: Shifting Vision Processing Closer to the Sensor. *2015 International Symposium on Computer Architecture (ISCA)* (June 2015), 92–104.

[14] Vijay Janapa Reddi et al. 2019. MLPerf Inference Benchmark. *ArXiv: 1911.02549 (2019)* (Dec. 2019).

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv: 1512.03385v1* (2015).

[16] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv: 1704.04861 (2017)* (April 2017).

[17] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size. *arXiv preprint: 1611.10012* (2016).

[18] Norman P. Jouppi et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *44th Int'l Symp. on Computer Architecture (ISCA)*. 1–12.

[19] Tushar Krishna, Hyoukjun Kwon, Angshuman Parashar, Michael Pellauer, and Ananda Samajdar. 2020. *Data Orchestration in Deep Learning Accelerators*. 1–164.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. *International Conf. on Neural Information Processing Systems (NIPS)* (Dec. 2012), 1106–1114.

[21] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. 2019. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. *International Symposium on Microarchitecture (MICRO)* (Oct. 2019), 754–768.

[22] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2017. Rethinking NoCs for spatial neural network accelerators. In *2017 Eleventh IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*. 1–8.

[23] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. 2018. MAERI: Enabling Flexible Dataflow Mapping over DNN Accelerators via Reconfigurable Interconnects. *International Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (March 2018), 461–475.

[24] T.T. Lee and Soung-Yue Liew. 1996. Parallel routing algorithms in Benes-Clos networks. *IEEE INFOCOM '96. Conference on Computer Communications* (March 1996).

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2015. SSD: Single Shot MultiBox Detector. *arXiv preprint arXiv: 1512.02325v5 (2015)* (Dec. 2015).

[26] Tao Luo, Shaoli Liu, Ling Li, Yuqing Wang, Shijin Zhang, Tianshi Chen, Zhiwei Xu, Olivier Temam, and Yunji Chen. 2017. DaDianNao: A Neural Network Supercomputer. *IEEE Trans. Comput.* 66, 1 (Jan. 2017), 73–88.

[27] Francisco Muñoz Matrínez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna. 2021. STONNE: Enabling Cycle-Level Microarchitectural Simulation for DNN Inference Accelerators. *IEEE Computer Architecture Letters* 01 (July 2021).

[28] Francisco Muñoz-Martinez, José L. Abellán, Manuel E. Acacio, and Tushar Krishna. 2021. A Novel Network Fabric for Efficient Spatio-Temporal Reduction in Flexible DNN Accelerators. In *To appear in 15th IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*.

[29] Eric Qin, Ananda Samajdar, Hyoukjun Kwon, Vineet Nadella, Sudarshan Srinivasan, Dipankar Das, Bharat Kaul, and Tushar Krishna. 2020. SIGMA: A Sparse and Irregular GEMM Accelerator with Flexible Interconnects for DNN Training. *IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (March 2020).

[30] Karen Simonyan and Andrew Zisserman. 2016. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint: 1409.1556v6* (2016).

[31] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel Emer. 2017. Efficient Processing of Deep Neural Networks: A Tutorial and Survey. *arXiv preprint arXiv: 1703.09039v2 (2017)* (Aug. 2017).

[32] Zhongyuan Zhao, Hyoukjun Kwon, Sachit Kuhar, Weiguang Sheng, Zhigang Mao, and Tushar Krishna. 2019. mRNA: Enabling Efficient Mapping Space Exploration for a Reconfigurable Neural Accelerator. *International Symposium on Performance Analysis of Systems and Software (ISPASS)* (April 2019), 282–292.