electronics

MDPI

Article

Nebula: A Scalable and Flexible Accelerator for DNN Multi-Branch Blocks on Embedded Systems

Dawei Yang [1], Xinlei Li [2], Lizhe Qi [1,*], Wenqiang Zhang [1,*] and Zhe Jiang [3]

[1] Academy for Engineering and Technology, Fudan University, Shanghai 200433, China; dwyang18@fudan.edu.cn
[2] School of Statistics and Information, Shanghai University of International Business and Economics, Shanghai 201620, China; lixinlei@suibe.edu.cn
[3] Computer Science, University of Cambridge, Cambridge CB3 0FD, UK; zhejiang.uk@gmail.com
* Correspondence: 19110860064@fudan.edu.cn (L.Q.); 18110860061@fudan.edu.cn (W.Z.)

Abstract: Deep neural networks (DNNs) are widely used in many artificial intelligence applications; many specialized DNN-inference accelerators have been proposed. However, existing DNN accelerators rely heavily on certain types of DNN operations (such as Conv, FC, and ReLU, etc.), which are either less used or likely to become out of date in future, posing challenges of flexibility and compatibility to existing work. This paper designs a flexible DNN accelerator from a more generic perspective rather than speeding up certain types of DNN operations. Our proposed *Nebula* exploits the width property of DNNs and gains a significant improvement in system throughput and energy efficiency over multi-branch architectures. *Nebula* is a first-of-its-kind framework for multi-branch DNNs.

Keywords: DNN accelerators; multi-branch network; energy-efficient accelerators

Citation: Yang, D.; Li, X.; Qi, L.; Zhang, W.; Jiang, Z. *Nebula*: A Scalable and Flexible Accelerator for DNN Multi-Branch Blocks on Embedded Systems. *Electronics* **2022**, *11*, 505. https://doi.org/10.3390/electronics11040505

Academic Editors: Thierry A. Meynard and Jaime W. Zapata

Received: 17 January 2022
Accepted: 4 February 2022
Published: 9 February 2022

1. Introduction

Deep neural networks (DNNs) have been widely used for solving complex problems across a wide range of domains, including computer vision, speech processing, and robotics [1–4], while DNNs can achieve remarkable results on high-performance cloud-servers, it is still expected to perform efficiently when used locally on mobile/embedded devices, due to connectivity and latency limitations, as well as privacy and security concerns [5,6]. Since mobile devices have tight latency, throughput, and energy constraints, many specialized DNN-inference accelerators, which achieve compelling results compared to traditional CPUs and GPUs, have been proposed [7]. DianNao [8] is one of the earliest accelerators designed for large-scale DNNs, followed by those in references [9–11]. These pioneers focus on the basic operations of early DNNs (such as Conv, FC, Pooling, and Activation). With the development of DNNs, various operations have been proposed and have been gradually supported by specialized DNN accelerators. Examples include Lee et al. (2019), who propose an efficient hardware architecture to accelerate ReLU by skipping zero activations, as ReLU sets negative values and leaves zero sparsity [12]; Ham et al. (2020) [13] apply a software–hardware co-design to achieve orders of magnitude energy efficiency improvements over Attention (one of the most important recent advancements in DNNs [14]); and, Ding et al. (2019) use double-buffering-based memory channels to improve the system throughput and performance in Depthwise Conv [15].

Despite their successes, these DNN accelerators rely heavily on specific types of DNN operations, as mentioned above, some of which might be less used or will become out of date in the future. For example, PReLU [16] and Swish [17] tend to work better than ReLU accross a number of challenging datasets. Moreover, the wide range of emerging operations (e.g., ViTransformer [18], Swin transformer [19]) poses a challenge of flexibility and compatibility to DNN accelerators. The above findings inspired us to design a flexible DNN accelerator from a more generic perspective rather than speeding up specific DNN

operations. The architecture of a DNN is often specified by: width and depth: the depth is defined as the number of layers, and a deep depth leads the DNN to a better generalization due to the learned features at various levels of abstraction; while the width is defined to be the maximum number of branches inside a block (or a layer, see Figure 1 for more details), which captures salient features adaptively in different branches. It is difficult to use the depth property in hardware acceleration because the dataflow of a DNN inference is in the form of a chain and has a strict *dependency* between two layers. For this reason, we take a closer look at the width property: the multi-branch topology (e.g., Figure 1) is associated with high similarity and independence, which provides an enormous potential for hardware acceleration.

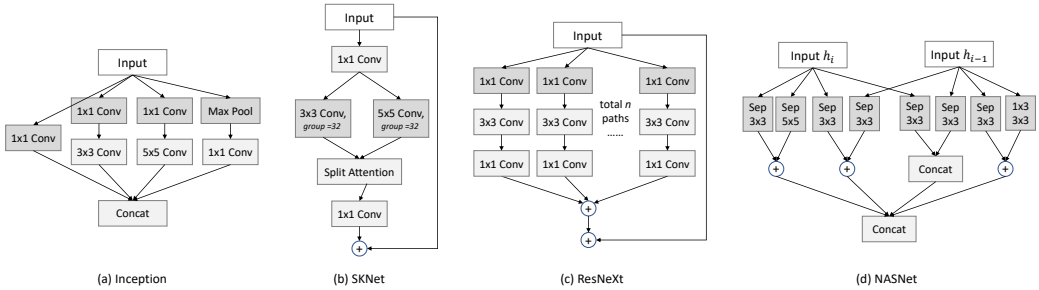


Figure 1. Multi-branch architectures.

Research Challenge: The existing research has not always recognized the importance of width, where the multi-branch architectures can only be executed in a *sequential* or *partitioned* manner.

Sequential execution is an essential yet naive method to run multiple branches inside DNN blocks on an accelerator, e.g., references [20,21]. This method sees a multi-branch block as multiple independent branches and then executes them sequentially (shown in Figure 2a). Therefore, sequential execution forbids data reuse between branches, which brings in redundant memory accesses of the same input data. The frequent redundant memory accesses result in significant reductions in *system throughput* and *energy efficiency*. More seriously, the weakness is magnified when the number of branches or channels increases. Considering a two-branch block taking a 128-channel feature map as input, the sequential mode still has to access the external memory 256 (2×128) times to fetch the input data.

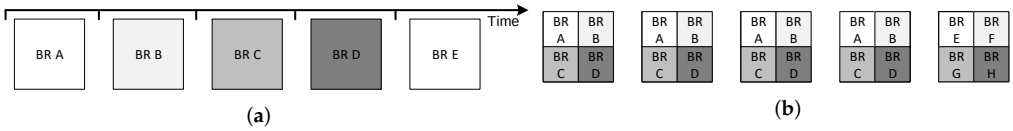


Figure 2. Executing multiple branches on conventional accelerators. (**a**) Sequential execution (BR: branch); (**b**) Partitioned execution (BR: branch).

Different from sequential execution, some of the flexible accelerators, e.g., references [8,9], enable the *partitioned execution* of multi-branch blocks. That is, the accelerator distributes the branches into different portions of the hardware and then executes them in parallel. Ideally, partitioned execution can alleviate the issues introduced in sequential execution by enabling a certain level of data reuse between branches. However, these accelerators were not designed for a multi-branch architecture, ignoring the branches' inner connection and executing them *independently*. Moreover, it is very difficult to ensure that branches fetch the same data input at the same time point, as different branches usually involve uneven computation complexity. Compared to sequential execution, partitioned execution also needs to solve other system-level problems, e.g., the workload balance between processors. Potential solutions to these problems (e.g., task migration) usually involve

additional software complexity and overhead, which further magnifies the aforementioned issues.

In summary, existing accelerators pay little or no attention to multi-branch architectures. Although a single multi-branch block may only have limited impacts to the system performance, there are often 10s or 100s of multi-branch blocks stacked in a DNN model, so the cumulative impact will seriously affect the throughput and energy efficiency of the entire system.

Contributions: In this paper, we introduce *Nebula*, the first DNN accelerator to simultaneously guarantee the system throughput and energy efficiency of multi-branch blocks. To achieve this, we present:

- A fully scalable and configurable *hardware architecture* for DNN execution, containing fully-connected network-on-chip (NoC), hierarchical memory meshes, and three transaction modes. The hardware architecture provides the flexibility to optimize data reuse and transaction flows between the branches;
- A *compiler* to map multi-branch DNN blocks on the proposed hardware, which firstly *decomposes* the high-dimensional operations (such as Conv, Table 1) down into channel-level primitives and then maps them to specified processors via a *2-level mapping*. This procedure optimizes data flow and eliminates repetitive data transfers, which effectively improves the efficiency of parallel computation and the workload balance of the processors. Furthermore, by maximizing the parallelism of multi-branch processing, the novel data flow also significantly increases the throughput and workload balance of the system. *Nebula* hence achieves considerable improvements in system throughput and energy efficiency with respect to conventional DNN accelerators;
- Comprehensive experiments to examine resource consumption, energy efficiency, scalability, and system-level throughput against state-of-the-art DNN accelerators.

The rest of this paper is organized as follows: Section 2 describes the basic concepts of DNN. Sections 3 and 4 detail the design of *Nebula*, followed by the evaluation given in Section 5. Section 6 makes conclusions.

Table 1. Shape parameters of Conv.

Parameter	Description
B	number of branches
H/W	input feature map height/width
M/N	number of input/output channels
K	kernel (filter) size
kw/kh	kernel (filter) width/height

2. Preliminaries: Width of DNN

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Models are trained by using a large set of labeled data and neural network architectures that contain many layers. Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as deep neural networks. As its name implies, one of the primary characteristics of DNNs is their depth. The 'depth' can be defined as the longest path between an input neuron and an output neuron. Traditional neural networks only contain two to three hidden layers, while deep networks can easily contain over 10s or 100s of layers. However, is a great depth always necessary? The depth is an important issue to consider because great depth does not come without downsides. For example, a deeper network results in increased sequential processing and delay; it is also more difficult to

parallelise and is, therefore, less appropriate for applications that require rapid response times.

The concept of width in DNN was first introduced in Inception [22], where multiple kernels of different sizes were implemented within the same layer. To be more specific, the width refers to the number of branches within a DNN block that can be presented as:

$$\mathcal{F}_{stm}(x) = \mathcal{M}(\mathcal{T}_1(x), \mathcal{T}_2(x), \ldots, \mathcal{T}_c(x))$$

where x is the input of the block; $\mathcal{M}$ can be any arbitrary function (such as weighted sum, concatenation, or element-wise addition) to merge the results of the set of transformation $\mathcal{T}$; c is the number of $\mathcal{T}$; and $\mathcal{T}_i$ can be a different set of operations (e.g., Conv, FC, Activation, etc.) on the ith branch. Taking the Inception block (Figure 1a) as an example, the input is fed into four branches. The first branch of the Inception block has one layer of 3×3 convolution, while the second to fourth branches stack two layers together. At the end of the block, the results from each branch are merged by Concat.

Instead of simply going deeper in terms of the number of layers, Inception goes wider and increases the number of branches with different kernel sizes. The idea is intuitive and it has been recognized in the neuroscience community: the receptive field-size of human visual cortical neurons is modulated by the stimulus according to the size of the target. With a multi-branch architecture, Inception can adaptively capture salient features in different branches. By contrast, ResNet builds very deep networks and solves two tricky problems—vanishing gradient and degradation; however, we must be aware that the residual block of ResNet [23] is also a special form of multi-branch architecture with two branches (the original path and the identity skip path). Experiments [24–26] show that increasing DNN width is a more effective way of gaining accuracy than going deeper, especially when depth starts to give diminishing returns for existing models. As shown in Figure 1c, ResNeXt, one of the most powerful DNN backbones to date, further expands the number of branches to 32. In addition to expert-designed DNNs, the effectiveness of multi-branch is also recognized by AutoML, a project released by Google, which adopts the neural architecture search algorithm to search for the best neural network (Figure 1d).

3. *Nebula*: Hardware Architecture

In this section, we introduce the *Nebula* hardware architecture (Figure 3). The *Nebula* hardware design partitions and groups essential components as *cluster*s, keeping scalability and flexibility in mind. To customize the hardware for different-sized networks, the system designer only needs to add/remove a specific number of clusters to/from the existing architecture.

3.1. Cluster

Figure 3b illustrates the top-level design of a cluster. As shown, a cluster contains three key components: a *cluster router*, a *PE cluster*, and *cluster RAMs*, introduced below.

Cluster Router: In a cluster, a cluster router physically connects the PE cluster and cluster RAMs, which provides data fetch interfaces for the PEs. At the same time, the cluster router also connects to the cluster routers in the other clusters, which enables inter-cluster communication. We established these connections on three communication channels, designed for the transactions of *input feature maps* (*Ifmaps*, 32 bits), *parameters* (32 bits), and *output feature maps* (*Ofmaps*, 64 bits), respectively.

PE Cluster: A Processing Element (PE) cluster contains 9 multiply-and-accumulate (MAC) PEs (see Figure 4) arranged in a 3×3 array. These PEs are connected to the routers (namely *PE router*s) of a 2D mesh-type open-source NoC [27].

The PE routers contain the same communication channels as the cluster routers. With the connections between the PE routers and cluster routers, a PE can directly communicate with any other PE, either in the same or a different cluster, without memory access.

The design of the PE keeps resource efficiency in mind. Therefore, a PE only contains four key elements (see Figure 4): an interpreter, FIFO queues, an adder, and a multiplier. The

interpreter receives transactions from a PE router and converts them to the corresponding MAC operations. The FIFO queues behave as the scratchpads and buffer MAC operations. The adder and the multiplier take charge of the calculations in the executed networks, i.e., addition and multiplication.

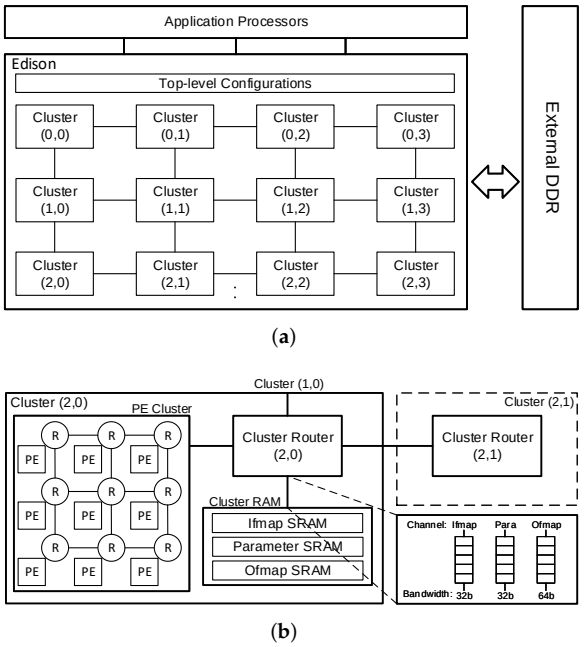


Figure 3. Hardware architecture. (**a**) Hardware architecture: top level; (**b**) Hardware architecture: cluster (2, 0). R: PE router.

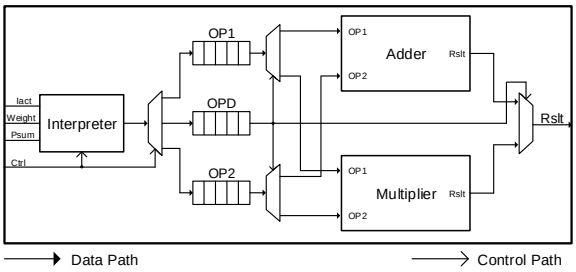


Figure 4. Micro-architecture of a MAC PE. (OP: operator; OPD: oprand; Rslt: result).

Cluster RAM: Each cluster also contains a group of static RAM (SRAM) banks, namely *cluster RAM*, for storing the run-time data for the PEs. Compared to the conventional DNN accelerators (e.g., references [8,21]), in which all PEs access a globally shared external memory, the partitioned memory effectively reduces transaction delay and resource contention during memory accesses, since most of the data fetch from the PEs can be handled within the same cluster. This partitioned memory hierarchy considerably reduces the power and time consumption caused by memory accesses and contentions in the conventional DNN accelerators. Each cluster RAM has three groups of RAM banks, storing Ifmaps, parameters, and Ofmaps. These banks are connected to the corresponding communication channels of the cluster router.

3.2. Other Elements

As shown in Figure 3a, as well as the clusters, the *Nebula* hardware also contains another two key elements: the top-level configurations and the external Dynamic RAM (DRAM). The top-level configuration regulates the run-time data flow between the clusters (discussed in Section 4) and globally shared data, e.g., the Ifmaps in the first layer and the Ofmaps in the final layer.

3.3. Transaction Modes

The proposed hardware supports three transaction modes, which can be selectively activated at run-time.

Unicast/anycast (Figure 5a): Each SRAM bank transfers data independently to a PE. This transaction mode maximizes the hardware bandwidth when data reuse is unavailable.

Broadcast (Figure 5b): An SRAM bank delivers data to all PEs. This transaction mode exploits data reuse in any spatial reuse patterns. It can reduce power consumption caused by a large data movement (e.g., Ifmaps).

Multicast (Figure 5c): An SRAM bank delivers data to PEs in the same cluster. Different from the broadcast, this transaction mode introduces more flexibility, but less data reuse and power consumption.

The proposed hardware design and the operating modes in *Nebula* provide scalability, configurability, and flexibility in the hardware, which allows *Nebula* to support the parallel computation of multi-branch blocks. The following section focuses on the compiler to map a network to *Nebula* hardware.

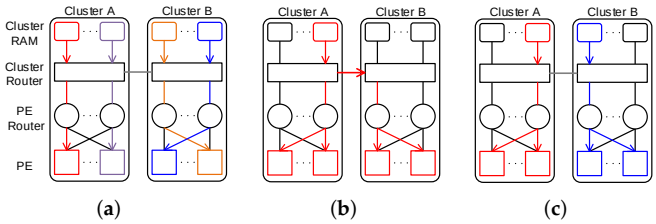


Figure 5. Three transaction modes supported by *Nebula*. (**a**) Unicast/anycast; (**b**) Broadcast; (**c**) Multicast.

4. *Nebula*: Compiler

Since CONV is usually more time-consuming and complex than other operations [9], in this section, we focus on the mapping of the CONV layers in *Nebula*.

To maximize the parallel computation and overall throughput of the execution, we introduce a three-step method:

Step 1: decomposition. The high-dimensional convolution is decomposed into 1D *Convolution Primitives* (*CP*s), which calculate the Psum results of a specified row in a channel. In the *same channel*, the CPs always have identical computation times, and can be executed in parallel.

Step 2: channel-level mapping. The CPs within the same channel are grouped and mapped to a set of *virtual PE*s (*vPE*s), which calculate and return the Ofmap of a specified channel. In the *same branch*, the vPE sets always have identical computational times, and can be executed in parallel.

Step 3: branch-level mappings. The vPE sets allocated for all branches are mapped to a set of *physical PE*s (*pPE*s). This procedure ensures load balance and identical computation times for all pPEs.

We detail the design of these three steps, below.

4.1. Decomposition

In decomposition, we split the computation of the CONV layer of each channel as a CP. Each CP generates one row of Psums by operating on one row of kernel weights and one row of Ifmaps. Figure 6 demonstrates an example of the first CP, where a 5×5 Ifmap is being filtered by a 3×3 kernel. Following this rule, when a $H \times W$ Ifmap is being filtered by a $kh \times kw$ kernel, we can decompose it to $kh \times (H - kh + 1)$ CPs.

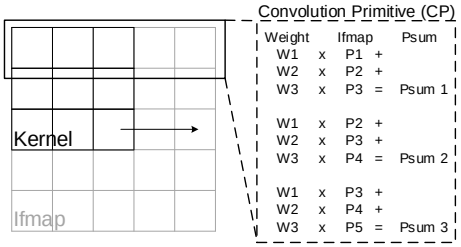


Figure 6. Decomposition of convolution primitives (CPs); (Ifmap size: 5×5; kernel size: 3×3).

4.2. Channel-Level Mapping

In channel-level mapping, we first allocate each CP to a vPE and then group the vPEs acquired from the same input channel as a vPE set. In each vPE set, we allocate the vPEs for the jth row of the kernel as the jth column of the vPE sets. Figure 7 illustrates this mapping strategy using the previously described example. The mapping ensures that the kernel weights can be reused across the vPEs horizontally, and the Ifmap pixels can be reused diagonally. After the channel-level mapping, for a branch with M input channels and N output channels, we can acquire $M \times N$ sets of vPEs.

The left-hand side of Figure 8 shows an example of the vPE sets created for the first CONV layer in a dual-branch network. In this example, the size of the Ifmap is 7×7 ($H = W = 7$); both branches have eight input channels ($M = 8$) and two output channels ($N = 2$); and the kernel sizes in these two branches are 3×3 and 5×5, respectively. After the channel-level mapping, it creates 16 ($M \times N$) sets of vPEs for each branch (32 in total). We denote the vPE set for the mth input channel and nth output channel in the bth branch as ***vPE_Set-b-n-m***. For instance, we denote the vPE sets for the third input channel and second output channel in the first branch as *vPE_Set-1-2-3*.

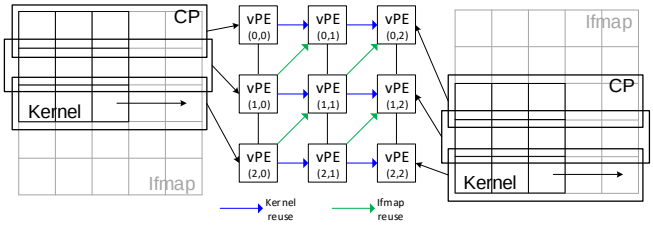


Figure 7. vPE set acquired in the channel-level mapping (Ifmap size: 5×5; kernel size: 3×3).

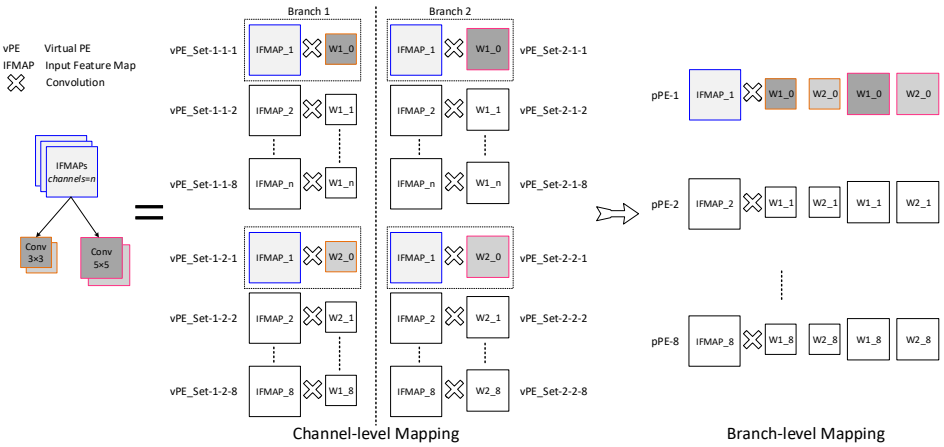


Figure 8. Channel-level mapping and branch-level mapping when $B = 2, M = 8$, and $N = 2$.

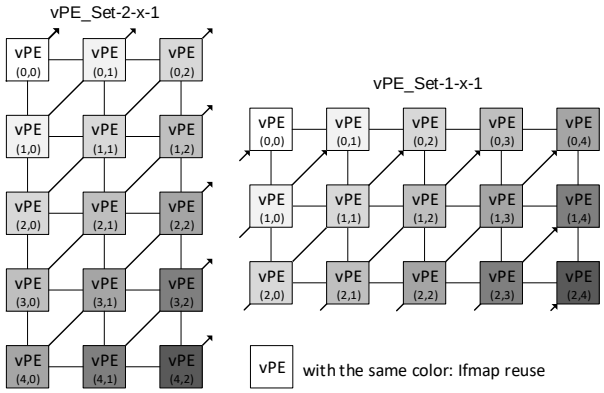


Figure 9. Data flow in pPE-1, using example illustrated in Figure 8.

4.3. Branch-Level Mapping

In branch-level mapping, we allocate and then execute the vPEs on the pPEs. The mapping of vPEs happens at the same granularity as the vPE set. This is because it preserves data reuse inside each vPE set (see Figure 7 and it provides flexibility to reuse the data between vPE sets. Moreover, the branch-level mapping strategy is proposed based on observations of the vPE sets acquired from the channel-level mapping, in which the vPE sets created for the same input channel (either in the same or different branches) always require the same Ifmap. The vPE sets created for the same branch always involve the same number of computations.With this in mind, branch-level mapping is proposed, targeting the following two objectives:

Maximizing Intra-pPE Data Reuse: Mapping tries to map vPE_Sets requiring the same Ifmap on the same pPE. Considering the example introduced in Figure 8, vPE_Set-1-1-**1** and vPE_Set-1-2-**1** require the same Ifmap pixel data, which hence can be mapped on the same pPE.

Balancing Inter-pPE Computation Load: Mapping tries to maintain the same number of vPE_Sets on all pPEs. Considering the aforementioned example, each pPE can be mapped by 4 vPE_Sets, including 2 vPE_Sets from each branch.

That is, the vPE_Set-1-1-**1**, 1-2-**1**, 2-1-**1**, and 2-2-**1** are mapped to pPE-**1**; and vPE_Set-1-1-**2**, 1-2-**2**, 2-1-**2**, and 2-2-**2** are mapped to pPE-**2**.

Different from channel-level mapping, always returning a fixed result for a given network, the results of the branch-level mapping vary depending on the availability of hardware resources. In the previously introduced example, if the system provides enough hardware resources, we only need to allocate 2 vPE_Sets to each pPE. In that case, vPE_Set-1-1-1 and 2-1-1 are mapped to pPE-1, and vPE_Set-1-2-1 and 2-2-1 are mapped to pPE-2.

With the three-step method, we can allocate multi-branch DNN blocks onto the *Nebula* hardware architecture. Additionally, we propose an optional optimization for the data flow in the system, maximizing overall energy efficiency, which is introduced in the following section.

4.4. Data Flow Optimization

We propose an optional optimization of the data flow from both intra- and inter-pPE perspectives:

Intra-pPE Data Flow: Channel-level mapping (Section 4.2) provides the diagonal reuse of the Ifmap pixels in each vPE set. At the same time, branch-level mapping (Section 4.3) ensures the vPE sets allocated to the same pPE require the same Ifmap. With this in mind, we optimize the intra-pPE data flow by allowing the Ifmap pixels to be transmitted diagonally, crossing the vPE set. Figure 9 demonstrates the optimized data flow for pPE-1 in the previously introduced example.

Inter-pPE Data Flow: As introduced in Section 3, the *Nebula* hardware architecture is partitioned and modularized. Hence, we try to allocate pPEs requiring the same data input

(e.g., Ifmaps) to the same cluster, and then use the *multicast* transaction mode (Figure 5c) during run-time to reduce the number of memory fetches.

Having described the *Nebula* hardware and compiler design above, in next section, we comprehensively examine the proposed system.

4.5. Discussion: Compatibility

Like much of the current research on DNN accelerators (e.g., references [8,9,28]) , the *Nebula* design mainly focuses on the CONV layers, since the CONV layers contribute more than 95% of the workload of the popular DNNs. The design of *Nebula* always keeps compatibility in mind. Therefore, the existing methods designed for the other layers can be added in *Nebula* to further accelerate DNN execution, e.g., reference [12] for ReLU acceleration. When we evaluated the *Nebula* in Section 5, we disabled the acceleration of other DNN layers to ensure the accurate evaluation of the paper's contribution and a fair comparison with the other baseline accelerators.

5. Evaluation

We now evalaute *Nebula* using comprehensive experiments.

Experimental Platform: We built the *Nebula* hardware on a Xilinx VC709 evaluation board. Specifically, the hardware was implemented using a BlueSpec System Verilog [29] and configured into two variants, named *Nebula* and *Nebula-H* (high performance), containing four and six *clusters* (72 and 108 *processors*), respectively.

For the baseline DNN accelerators (**BAs**), however, different variants were demonstrated in the previous work. As reviewed in Section 1, existing MCS frameworks usually execute DNN multi-branch blocks by either using sequential execution (**SEQ**) or partitioned execution (**PAR**). Therefore, we built two BSs (i.e., *BA-PAR* and *BA-SEQ*) using the methods introduced in references [8,9], respectively. All evaluated accelerators were executed at 150 Mhz and configured to run at full performance. It is important to note that the modularized design of *Nebula* (see Section 3) allows it to be configured with any number of clusters.

5.1. Hardware Overhead

Experimental Setup: We first evaluated the hardware consumption of a cluster in *Nebula* against two general-purpose processors (MicroBlaze (MB-F) and RISC-V), and then compared the two variants of *Nebula* with the two DNN accelerators (*BA-PAR* and *BA-SEQ*). MB-F was fully-featured, enabling five-stage pipeline, debug modules, etc. The RSIC-V was implemented based on reference [30], which supports all functionalities of the MB-F, as well as multi-branch, out-of-ordering processing, and other related functionalities (e.g., branch prediction, etc.). All components were synthesized and implemented using Vivado (2020.2), and compared using look-up tables (LUTs), registers, DSPs, and Block RAMs (BRAMs).

Observation 1: The design of *Nebula* is resource-efficient.

This observation is given in Table 2. As shown, a cluster (nine processors) in *Nebula* consumed similar resources compared to the general-purpose processors: MB-F (157.3% LUTs, 132.6% registers, 87.7% DPSs) and RSIC-V (103.9% LUTs, 35.61% registers, 92.3% DSPs). When compared to the other DNN accelerators, *Nebula* was also resource-efficient, as it involved less consumption of LUTs, registers, DSPs, and RAMs.

Observation 2: *Nebula-H* involves more memory consumption compared to conventional accelerators.

As shown in Table 2, *Nebula-H* consumed less LUTs and registers but more BRAMs compared to the conventional accelerators. This mainly resulted from the modularized design (see Section 3), which allocated dedicated memory resources to each cluster. These additionally consumed resources are intended to bring an enhancement to system throughput, which is specifically examined in the following sections.

Table 2. Hardware overhead (Implemented on FPGA).

	LUTs	Registers	DSP	RAM (KB)
MB-F	4908	4385	41	2048
RSIC-V	7432	16,321	39	2048
Nebula (Cluster)	7723	5813	36	2048
BA-PAR	70,560	141,120	360	13,824
BA-SEQ	89,773	70,121	733	16,920
Nebula	51,737	43,544	288	16,384
Nebula-H	71,015	57,443	432	24,576

5.2. Experiments Using Synthetic Workloads

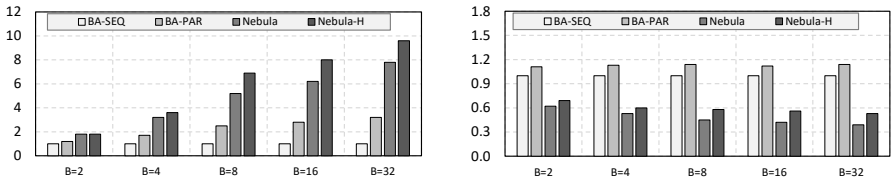
Experimental Setup: To evaluate the effectiveness of *Nebula* performing on multi-branch blocks against the existing DNN-inference accelerators, we carried out experiments on blocks with a different number of synthetic workloads, i.e., branches (B). Since convolutions account for over 90% of DNN operations and dominate the run-time [8,31], each branch was constructed with a convolution with a random kernel size. We performed the experiments 1000 times, and recorded the average throughput and run-time energy cost. For each experimental configuration, we normalized the results using *BA-SEQ*.

Observation 3: While executing the DNNs blocks, *Nebula* significantly outperforms the sequential and partitioned executions in both throughput and energy consumption.

This observation is illustrated in Figure 10. As shown, the benefits of introducing two-level mapping in *Nebula* (see Section 4) increased with the increment of B. In particular, when $B = 32$, the throughput speedup ratio reached 8.8× and the energy cost decreased to 39%. Based on this observation, we deducted that the benefit to mainstream DNN backbones would be increasingly significant, where B can be as high as 64 and 128.

Observation 4: *Nebula-H* outperformed *Nebula*, however it required more energy consumption.

As shown in Figure 10, we also reported that *Nebula-H* consistently outperformed *Nebula* in the performance speedup ratio but suffered from a certain amount of energy cost. This means that if a production desires further improvement but is willing to sacrifice a little energy, *Nebula-H* would be a better choice.



(**a**) (**b**)

Figure 10. Evaluating using synthetic workloads (normalized by *BA-SEQ* in each configuration). (**a**) Speedup ratio: throughput; (**b**) Energy cost.

5.3. Experiments Using Mainstream DNNs

Experimental Setup: We ran the same experiments as in Section 5.2 using mainstream DNNs (including NASNet, ResNeSt, and ResNeXt-34/50/101) and two typical DNN accelerators. We performed each test 20 times, and recorded the average throughput and run-time energy cost. For each experimental configuration, the experimental results were normalized by the results of *BA-SEQ*.

Observation 5: *Nebula* outperforms conventional DNN accelerators while executing the mainstream DNNs that have a high computational complexity due to their large depths and widths.

This observation is shown by Figure 11. Compared to the baseline DNN accelerators, *Nebula* consumed around 30% less run-time energy cost, but achieved much better throughput. Although speed and energy consumption have always been difficult to optimize at the same time, *Nebula* provides an elegant and effective solution because of its well-designed memory hierarchy and data flow, thereby reducing unnecessary data movement. Furthermore, we reported that the speedup ratio (5–6×) of *Nebula* here is weaker than Figure 11a. This is because DNN models also contain non-multi-branch structures. As highlighted in Section 4.5, we disabled the acceleration for these non-multi-branch structures to ensure a fair and accurate evaluation.

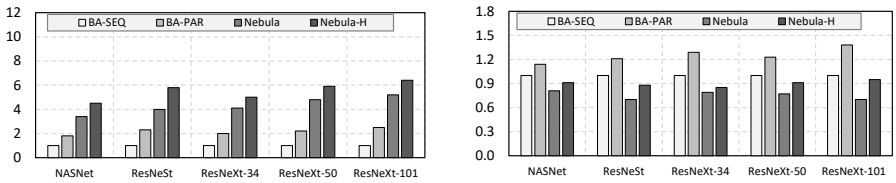


Figure 11. Evaluating using mainstream DNNs (normalized by *BA-SEQ* in each configuration). (**a**) Speedup ratio: throughput; (**b**) Energy cost.

5.4. Scalability

We acknowledge that the *scalability* finally determines the feasibility of the proposed design [32]. Therefore, we examined the scalability (Figure 12) of *Nebula* using a varying number of clusters.

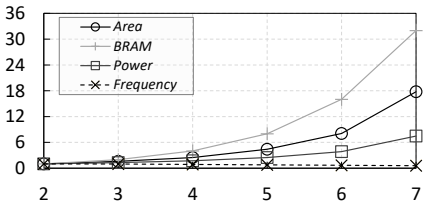


Figure 12. Scalability: the x-axis denotes the scaling factor η and the y-axis denotes the increase ratio compared with four-cluster *Nebula*.

Experimental Setup: We adopted the same method described in Section 5.1 to synthesize and implement *Nebula* with a scaling number of clusters [7]. In the experiments, we introduced a scaling factor: η, to control the number of clusters (2^{η}). The experimental results showed the scalability of the consumption of area, memory, and power, as well as maximum clock frequency. The results are normalized with the four-cluster *Nebula*.

Observation 6: *Nebula* provides good scalability in the consumption of area, BRAMs, power, and the maximum frequency.

As shown, the consumption of the BRAMs was linearly scaled with η, since we allocated a fixed amount of 2048 KB BRAMs for each cluster. Different from this, the ratio of the area increment was always below the linearly increased η. This benefited from the optimization of the synthesis. As expected, we also observed that the ratio of the power increment was lower than the increment of η. This is because the power consumption of a system is determined by four factors [33,34]: *voltage*, *clock frequency*, *toggle rate*, and *design area*. When the other factors are constant, the design area dominates the overall power consumption. Lastly, we reported a loss of the maximum frequency in *Nebula* when η increased. Such a decrease in the maximum frequency is acceptable, since these additionally introduced clusters bring additional computation capability (shown in Sections 5.2 and 5.3).

6. Conclusions

The development of DNNs brings challenges to DNN accelerators due to various emerging operations. In this paper, we identify the importance of the DNN width prop-

erty for hardware acceleration, and propose a new hardware architecture (*Nebula*) for DNN multi-branch blocks, including a hardware platform and a dedicated compiler. The hardware platform includes a fully-connected NoC, hierarchical memory meshes, and different transaction modes. This scalable and configurable hardware architecture enables the flexibility to optimize data reuse and transaction flows between most DNN layers, as well as multi-branch blocks. Additionally, the compiler decomposes the high-dimensional operations of multi-branch blocks into channel-level primitives and then maps them to specified PEs using a two-level mapping method. This novel data flow eliminates repetitive data movement, energy-hungry data staging, and transmission costs. Furthermore, by maximizing the parallelism of multi-branch processing, the data flow significantly enhances the throughput and workload balance of the system. Thus, *Nebula* achieves better performance on both speed and energy consumption when processing mainstream DNNs. In conclusion, *Nebula*, which exploits the width property of DNN, provides an elegant and effective solution to the problem of designing DNN hardware accelerators.

Author Contributions: Conceptualization, D.Y. and X.L.; methodology, D.Y. and Z.J.; software, W.Z. and Z.J.; validation, D.Y., X.L. and L.Q.; writing—original draft preparation, W.Z.; writing—review and editing, W.Z. and Z.J.; supervision, D.Y.; project administration. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by National Key R&D Program of China (2020AAA0108301), National Natural Science Foundation of China (No. 62072112), and the Scientific and Technological Innovation Action Plan of Shanghai Science and Technology Committee (No. 205111031020).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The authors would like to thank the editors and reviewers for their efforts and suggestions to improve our manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
2. Bengio, Y.; Goodfellow, I.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2017; Volume 1.
3. Yang, D.; Li, X.; Dai, X.; Zhang, R.; Qi, L.; Zhang, W.; Jiang, Z. All in one network for driver attention monitoring. In Proceedings of the ICASSP 2020—2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain, 4–8 May 2020; pp. 2258–2262.
4. Wang, H.; Zhu, Y.; Adam, H.; Yuille, A.; Chen, L.C. Max-deeplab: End-to-end panoptic segmentation with mask transformers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 19–25 June 2021; pp. 5463–5474.
5. Jiang, Z.; Zhao, S.; Wei, R.; Yang, D.; Paterson, R.; Guan, N.; Zhuang, Y.; Audsly, N. Bridging the Pragmatic Gaps for Mixed-Criticality Systems in the Automotive Industry. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2021**. [CrossRef]
6. Jiang, Z.; Zhao, S.; Dong, P.; Yang, D.; Wei, R.; Guan, N.; Audsley, N. Re-thinking mixed-criticality architecture for automotive industry. In Proceedings of the 2020 IEEE 38th International Conference on Computer Design (ICCD), Hartford, CT, USA, 18–21 October 2020; pp. 510–517.
7. Jiang, Z.; Audsley, N.C.; Dong, P. Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems. In Proceedings of the 2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Porto, Portugal, 11–13 April 2018; pp. 75–84.
8. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM Sigarch Comput. Archit. News* **2014**, *42*, 269–284. [CrossRef]
9. Chen, Y.H.; Krishna, T.; Emer, J.S.; Sze, V. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J.-Solid-State Circuits* **2016**, *52*, 127–138. [CrossRef]
10. Parashar, A.; Rhu, M.; Mukkara, A.; Puglielli, A.; Venkatesan, R.; Khailany, B.K.; Emer, J.S.; Keckler, S.W.; Dally, W.J. Scnn: An accelerator for compressed-sparse convolutional neural networks. *Comput. Archit. News* **2017**, *45*, 27–40. [CrossRef]

11. Sharma, H.; Park, J.; Mahajan, D.; Amaro, E.; Kim, J.K.; Shao, C.; Mishra, A.; Esmaeilzadeh, H. From high-level deep neural models to FPGAs. In Proceedings of the 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 15–19 October 2016; pp. 1–12.
12. Lee, G.; Park, H.; Kim, N.; Yu, J.; Jo, S.; Choi, K. Acceleration of DNN Backward Propagation by Selective Computation of Gradients. In Proceedings of the Annual Design Automation Conference, Las Vegas, NV, USA, 2–6 June 2019.
13. Ham, T.J.; Jung, S.J.; Kim, S.; Oh, Y.H.; Park, Y.; Song, Y.; Park, J.H.; Lee, S.; Park, K.; Lee, J.W.; et al. Aˆ 3: Accelerating Attention Mechanisms in Neural Networks with Approximation. In Proceedings of the 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA), San Diego, CA, USA, 22–26 February 2020.
14. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In Proceedings of the 2017 31st Conference on Neural Information Processing Systems (NIPS), Long Beach, CA, USA, 4–9 December 2017 ; pp. 5998–6008.
15. Ding, W.; Huang, Z.; Huang, Z.; Tian, L.; Wang, H.; Feng, S. Designing efficient accelerator of depthwise separable convolutional neural network on FPGA. *J. Syst. Archit.* **2019**, *97*, 278–286. [CrossRef]
16. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Boston, MA, USA, 7–12 June 2015; pp. 1026–1034.
17. Ramachandran, P.; Zoph, B.; Le, Q.V. Searching for activation functions. *arXiv* **2017**, arXiv:1710.05941.
18. Kolesnikov, A.; Dosovitskiy, A.; Weissenborn, D.; Heigold, G.; Uszkoreit, J.; Beyer, L.; Minderer, M.; Dehghani, M.; Houlsby, N.; Gelly, S.; et al. An image is worth 16 $\times$ 16 words: Transformers for image recognition at scale. In Proceedings of the 2021 International Conference on Learning Representations (ICLR), Vienna, Austria, 3–7 May 2021.
19. Liu, Z.; Lin, Y.; Cao, Y.; Hu, H.; Wei, Y.; Zhang, Z.; Lin, S.; Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, Canada, 11–17 October 2021; pp. 10012–10022
20. Zhang, X.; Wang, J.; Zhu, C.; Lin, Y.; Xiong, J.; Hwu, W.M.; Chen, D. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018.
21. Xu, P.; Zhang, X.; Hao, C.; Zhao, Y.; Zhang, Y.; Wang, Y.; Li, C.; Guan, Z.; Chen, D.; Lin, Y. AutoDNNchip: An automated dnn chip predictor and builder for both FPGAs and ASICs. In Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA), Seaside, CA, USA, 23–25 February 2020.
22. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 7–12 June 2015; pp. 1–9.
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2016; pp. 770–778.
24. Li, X.; Wang, W.; Hu, X.; Yang, J. Selective kernel networks. In Proceedings of the 2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Long Beach, CA, USA, 16–20 June 2019; pp. 510–519.
25. Xie, S.; Girshick, R.; Dollár, P.; Tu, Z.; He, K. Aggregated residual transformations for deep neural networks. In Proceedings of the IEEE 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 22–25 July 2017; pp. 1492–1500.
26. Zhang, H.; Wu, C.; Zhang, Z.; Zhu, Y.; Zhang, Z.; Lin, H.; Sun, Y.; He, T.; Mueller, J.; Manmatha, R.; et al. Resnest: Split-attention networks. *arXiv* **2020**, arXiv:2004.08955.
27. Plumbridge, G.; Audsley, N. Blueshell: A platform for rapid prototyping of multiprocessor NoCs. *Comput. Archit. News* **2014**, *41*, 107–117. [CrossRef]
28. Jiang, Z.; Dai, X.; Audsley, N. HIART-MCS: High Resilience and Approximated Computing Architecture for Imprecise Mixed-Criticality Systems. In Proceedings of the 2021 IEEE Real-Time Systems Symposium (RTSS), Dortmund, Germany, 7–10 December 2021; pp. 290–303.
29. Bluespec System Verilog. Available online: https://bluespec.com (accessed on 12 January 2021).
30. Mashimo, S.; Fujita, A.; Matsuo, R.; Akaki, S.; Fukuda, A.; Koizumi, T.; Kadomoto, J.; Irie, H.; Goshima, M.; Inoue, K. An Open Source FPGA-Optimized Out-of-Order RISC-V Soft Processor. In Proceedings of the 2019 International Conference on Field-Programmable Technology (ICFPT), Tianjin, China, 9–13 December 2019.
31. Cong, J.; Xiao, B. Minimizing computation in convolutional neural networks. In *International Conference on Artificial Neural Networks*; Springer: Cham, Switzerland , 2014.
32. Jiang, Z.; Dai, X.; Dong, P.; Wei, R.; Yang, D.; Audsley, N.; Guan, N. Towards an Analysable, Scalable, Energy-Efficient I/O Virtualization for Mixed-Criticality Systems. *IEEE Trans.-Comput.-Aided Des. Integr. Circuits Syst.* **2021**. [CrossRef]
33. Bellaouar, A.; Elmasry, M. *Low-Power Digital VLSI Design: Circuits and Systems*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
34. Jiang, Z.; Yang, K.; Fisher, N.; Audsley, N.; Dong, Z. Pythia-MCS: Enabling Quarter-Clairvoyance in I/O-Driven Mixed-Criticality Systems. In Proceedings of the 2020 IEEE Real-Time Systems Symposium (RTSS), Houston, TX, USA, 1–4 December 2020; pp. 38–50.