

---

# LOCAL: LOW-COMPLEX MAPPING ALGORITHM FOR SPATIAL DNN ACCELERATORS

---

A PREPRINT

**Midia Reshadi\***

School of Computer Science and Statistics  
Lero, Trinity College Dublin  
Dublin 2, Ireland  
Midia.Reshadi@tcd.ie

**David.Gregg**

School of Computer Science and Statistics  
Lero, Trinity College Dublin  
Dublin 2, Ireland  
David.Gregg@tcd.ie

November 8, 2022

## ABSTRACT

Deep neural networks are a promising solution for applications that solve problems based on learning data sets. DNN accelerators solve the processing bottleneck as a domain-specific processor. Like other hardware solutions, there must be exact compatibility between the accelerator and other software components, especially the compiler. This paper presents a LOCAL (Low Complexity mapping Algorithm) that is favorable to use at the compiler level to perform mapping operations in one pass with low computation time and energy consumption. We first introduce a formal definition of the design space in order to define the problem's scope, and then we describe the concept of the LOCAL algorithm. The simulation results show  $2\times$  to  $38\times$  improvements in execution time with lower energy consumption compared to previous proposed dataflow mechanisms.

**Keywords** Deep neural network · spatial DNN accelerators · mapping · low-complex algorithm.

## 1 Introduction

The growing amount of data brought learning-based systems from dreams to reality [1]. Among machine learning methods, deep neural networks achieved better results; therefore, they have attracted much attention from both research and industry [2],[3],[4]. Deep neural networks are widely used today in many applications such as self driving cars [5], recommender systems [6], and language translation [7]. Due to performance bottleneck and high energy consumption of processing all parts of DNN at the software level, domain-specific hardware, also called DNN accelerators, has been proposed in both resource constraints devices such as IoT edge [8] and high performance cloud servers [3].

The design space of deep neural network accelerators comprises hardware resource and data mapping strategy [2]. The hardware resources generally consist of multi-level storage hierarchy to exploit data reuse [9] and an array of processing elements (PE) to perform parallel computations. Each processing element, as shown in Fig. 1, comprises multiply and accumulate (MAC) logic connecting to local scratch pad memory, and all PEs are connected through the network-on-chip (NoC) interconnection.

Most of the proposed reference architectures [2][10][11][12] are different in their NoC topology and PE to memory connection. For instance, MAERI [13] is based on two binary fat trees, and Eyeriss [2] and Google TPU [3] use 2D grid-style topology. In some accelerators, the PE array is connected to only one internal scratchpad memory and, in others, to multiple memory banks.

Another point in the DNN accelerator's design is the data mapping strategy that deals with two important tasks: ❶ Staging data on the HW resources and, ❷ operation scheduling (Fig. 1). In other words, data mapping answers two

---

\*This paper is published in: 2021 IEEE Nordic Circuits and Systems Conference (NorCAS), with DOI: 10.1109/NorCAS53631.2021.9599862.

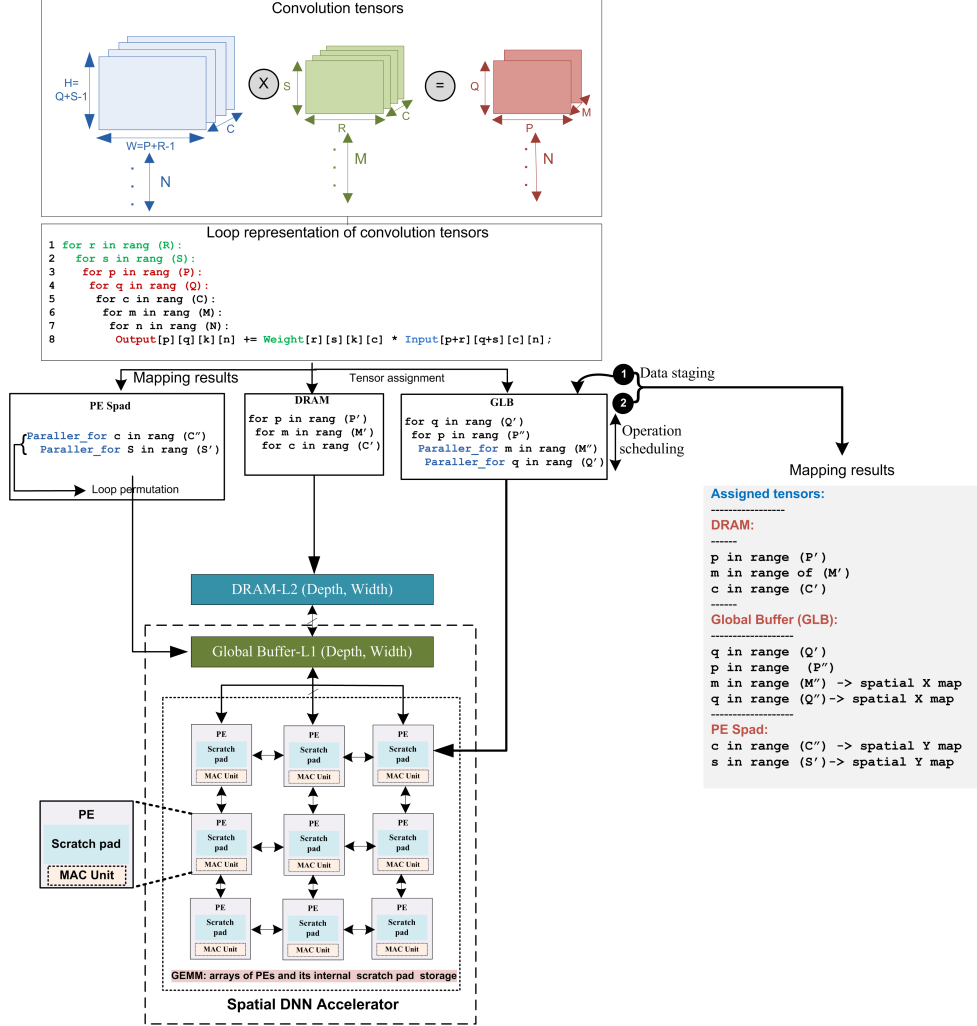


Figure 1: The overall concept of convolution tensors, their loop representation, and the mapping results of assigned tensors to memory elements including DRAM, GLB, and PE Spad of spatial DNN accelerator.

fundamental questions, where data be placed? And when are operations performed? Fig. 1 shows the concept of the mapping on spatial DNN accelerator.

The basic mapping methods [2] called *dataflow* [14] are based on reusing one of the primary convolution tensors, such as filter weights, input, or output activations. For instance, *NVDLA* [4] employs a *weight stationary* strategy to reuse weights, *Eyeriss* [2] uses *row stationary*, and *ShiDianNao* [15] employs *output stationary*.

Mapping is a more advanced concept than dataflow. It is about reusing multiple input values with specific ranges based on the shapes of DNN, dimensions of PE-arrays, levels and size of storage elements. Recently proposed mapping methods [16], [17] have shown that, beyond the reuse of only one parameter and also correct placement of several parameters, more energy efficiency will be gained compared with conventional dataflow methods.

However, the vital issue in mapping is the vast search space of the solutions and complexity as far as we face  $O(10^8)$  cases only in the fifth layer of VGG02 on Eyeriss accelerator that needs about 48 hours to find an optimal mapping based on an exhaustive brute-force search method. This number of mapping choices can rise to  $O(10^{72})$  for the 52-layer MobileNet-V2 [18][19].

Typically, the mapping strategy is deployed in the design-phase [20][21] or compile-time [22]. Hence, finding the optimal mapping in the *design-time* is essential but critical when it needs to be solved in *compile-time*. Several mapping algorithms have been proposed using evolutionary or learning-based algorithms to solve a problem [18][19], although, applying them at the compiler level extends the compile time due to many running iterations.

To tackle the complexity and reducing mapping time, we propose a mapping algorithm called *LOCAL*. *LOCAL*'s primary goal is finding a close-to optimal mapping in low computation time with admissible energy consumption.

We formally describe the problem to reach a clear and structured solution then we present the *LOCAL* mapping algorithm. Finally, we contrast our approach against other dataflow mechanisms, including weight, output, and row stationary in NVDLA, ShiDianNao, and Eyeriss accelerators, respectively.

Therefore, the main contributions of this paper are as follows:

- **Problem formulation.** We formulate the problem to achieve an accurate definition of the scope of the problem. Problem formulation defines the scope and dimension of the problem, so that helps to find an optimal solution between massive solution options.
- **A low-complexity one-pass mapping algorithm.** We propose a low complexity mapping algorithm called *LOCAL* with  $2\times$  to  $38\times$  higher speed compared to other dataflow mechanisms.
- **Usability at the compiler level.** *LOCAL* can perform a fast mapping at runtime in a single pass, making it a favorable option for compiler-level implementation.
- **Low energy consumption.** The second goal of *LOCAL* is reducing energy consumption. Simulation results show *LOCAL* is more energy-efficient than conventional dataflow techniques.

The paper is organized as follows: Section II provides preliminaries on convolution and spatial DNN accelerator; Section III presents the paper's motivation; Section IV presents problem formulation; Section V describes *LOCAL* mapping algorithm in detail; Section VI presents comprehensive evaluations; Section VII presents related work and Section VIII concludes the idea.

## 2 Preliminaries

The main assumptions of this paper are DNN and accelerator, and the final goal is the mapping algorithm that accepts those assumptions as inputs. This section introduces the basic definitions of convolution in modern DNNs and the structure of spatial DNN accelerators.

### 2.1 Convolutions

*Definition 2.1:* A convolution tensor (CT) comprises three tensors: filter weights, input and output feature maps. Hence, we define CT as:

$$CT = \{Weight, Input, Output\} \quad (1)$$

which

$$CT \in \mathbb{R}^{dimension} \quad (2)$$

and, dimensions are:

$$dimension = \{N, M, C, R, S, W, H, P, Q\} \quad (3)$$

which,  $W \in \mathbb{R}^{MCRS}$ ,  $I \in \mathbb{R}^{NCHW}$ , and  $O \in \mathbb{R}^{NMPQ}$ , are filter weights, input and output feature maps, respectively.

$$ct_i \in CT \quad (4)$$

$$CT = \{ct_1, ct_2, ct_3\} \quad (5)$$

$$CT = \{\mathbb{R}^{MCRS}, \mathbb{R}^{NCHW}, \mathbb{R}^{NMPQ}\} \quad (6)$$

We specify a shape of tensor as  $r^x \in tc_i$  that is:

$$r^x \in \mathbb{R}^y \quad x, y \in dimension \quad (7)$$

For example:

$$r^c \in Weight \rightarrow r^c \in \mathbb{R}^{MCRS} \quad (8)$$

We may summarize that, various CNNs have different tensor dimensions. In the software layer each tensor is defined as a loop-nests; for example, the related loop of  $r^c$  is:

$$r^c \rightarrow \text{for } c \text{ in range}(C) \quad (9)$$

For better illustration, Fig. 1 depicts the loop representation and matrix shape of convolution tensors. The mapping result is also shown in Fig. 1 which are mapped tensors to the storage elements of an accelerator.

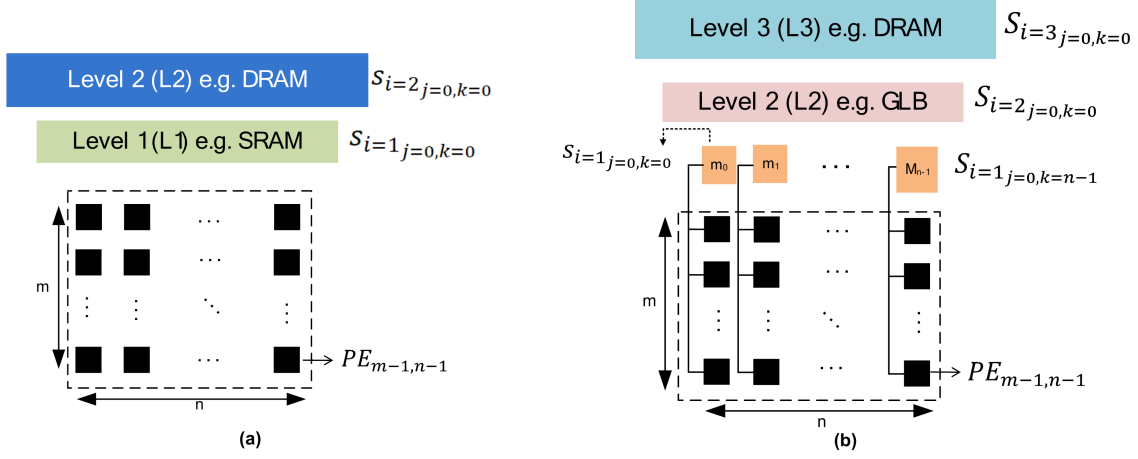


Figure 2: (a) *NVDLA-style*: 1D memory array and a 2D array of processing elements (PE) in  $(m, n)$  dimensions. (b) *Eyeriss-style*: 1D memory in each level, multiple memory elements in level 1  $(1, n)$ , and 2D arrays of PEs in  $(m, n)$ .

## 2.2 Spatial DNN accelerator

**Definition 2.2:** A spatial DNN accelerator (SPA) comprises an array of multi-level storage elements with hierarchy and array of processing elements:

$$SPA = \{Storage[i, j, k], PE[m, n]\} \quad (10)$$

Storage elements are:

$$s_{i,j,k}, \quad S \in \mathbb{R}^{i,j,k} \quad (11)$$

That  $i$  means the hierarchy level, and  $j$  and  $k$  are the indices of memory elements in a 2D space. Fig. 2(b), depicts an example of representing a memory element as  $s_{i=1, j=0, k=n-1}$ , that means the position of the memory element  $m_{n-1}$  in the 1<sup>st</sup> level of hierarchy in  $j = 0$  and  $k = n - 1$  dimension. For better readability, we present  $s[i, j, k]$  as  $s_{i,j,k}$ .

The size of each storage element  $s \in S$  at the  $i^{th}$  level is equal to:

$$|s| = Depth \times Width \quad (12)$$

Meanwhile, the array of processing elements is defined as:

$$PE_{m,n} \in \mathbb{R}^{x,y} \quad (13)$$

which, all  $(m \times n)$  PEs are connected by a Network-on-Chip (NoC).

Based on the connection between global memory and a array of processing elements, we consider two different accelerator types.

The first architecture, called *NVDLA-style*, comprises two internal memory elements ( $L0$  registers at PE and global buffer as  $L1$ ). The second architecture, called *Eyeriss-style*, includes three internal memory levels, including  $L0$  at PE, 1D array of multiple elements as  $L1$ , and global buffer as  $L2$ . The purpose of using  $L1$  memory elements is to connect each  $m_k$  elements with the range of  $k = 0 \rightarrow n - 1$  to the column of  $PE_{i,j}$  in the range of  $i = 0 \rightarrow m - 1$  and  $j = 0 \rightarrow n - 1$ . Therefore, we can summarize the memory to PE connections as follows:

*NVDLA-style* (Fig. 2(a)):

Level 1 (L1):  $s_{i=1, j=0, k=0}$

$$s_{i=1, j=0, k=0} \text{ connects to } PE_{i=0 \rightarrow m-1, j=0 \rightarrow n-1} \quad (14)$$

*Eyeriss-style* (Fig. 2(b)):

Level 1 (L1):  $s_{i=1, j=0, k=0 \rightarrow (m-1)}$

$$s_{i=1, j=0, k=0} \text{ connects to } PE_{m=0 \rightarrow (m-1), n=0} \quad (15)$$

$$s_{i=1, j=0, k=n-1} \text{ connects to } PE_{m=0 \rightarrow (m-1), n=(n-1)} \quad (16)$$

Fig. 2 shows the summarized and abstract structure of the spatial DNN accelerator. We represent the structural features of Eyeriss and NVDLA-style accelerators to achieve a clear problem definition.

### 2.3 Mapping Algorithm

The mapping of the convolution tensors (CT) onto the spatial DNN accelerator (SPA) is defined by the *mapping* function, including the following operations:

1. **Assignment.** Tensor assignment means assigning tensors to storage elements:

$$ct_i \in CT \text{ assign to } s_{i,j,k} \in S \quad (17)$$

2. **Bounding.** Bounding is the process of limiting tensor dimensions so that they are smaller than the size of storage elements:

$$|CT| \leq |S| \quad (18)$$

Hence, the tensor assignment is bounded by a specific range:

$$ct_i[0, range] \in CT \text{ assign to } s_{i,j,k} \in S \quad (19)$$

For example  $r^Q$  tensor with the following for-loop representation is assigned to  $s_{i=1,j=0,k=0}$  memory element at the 1<sup>st</sup> level of memory hierarchy:

$$\text{for } Q \text{ in } [0, 5) \text{ assigned to } s_{i=1,j=0,k=0}$$

3. **Scheduling.** Tensor order scheduling is the permutation of every allocated tensor. To be more precise, it involves defining the tensor order.

$$ct_i, ct_j, ct_k \in CT \text{ assign to } L_i \quad (20)$$

such that:

$$\begin{aligned} &ct_i[0, rang_i) \\ &ct_j[0, rang_j) \\ &ct_k[0, rang_k) \end{aligned}$$

Thus, the permutation is in the order of  $ct_i, ct_j, ct_k$ , respectively. For example, at the  $i^{th}$  level of memory, we may have the following order:

$$\begin{aligned} &\text{for } M \text{ in } [0, 5) \\ &\text{for } Q \text{ in } [0, 4) \\ &\text{for } P \text{ in } [0, 6) \end{aligned}$$

4. **Parallelization.** Spatial partitioning of assigned tensors is known as parallelization. More specifically, it refers to assigning a tensor to a subset of PEs in order to carry out parallel computation.

$$\text{Spatial computing} \rightarrow ct_i[0, rang) \quad (21)$$

which,  $ct_i[0, rang)$  is:

$$ct_i[0, rang) \in CT \text{ on } PE_{[i \text{ to } j]}(x|y) \in PEs \quad (22)$$

For example:

*Parallel\_for S in [0,7) on PE[0-7) Spatial X dimension*

Fig. 1 shows the example of final mapping results includes assigned, bounded, scheduled, and parallelized tensors to storage elements (DRAM, GLB, and PE Spad).

## 3 Motivation

To show the importance of the mapping strategy on energy consumption, we conducted an experiment generating 3,000 random mapping cases without any heuristics. We randomly map the fifth layer of VGG02 on the Eyeriss-style accelerator according to the configuration shown in Table 1. The results are classified into three categories, including *random\_max*, *random\_med*, and *random\_min*, as the cases with maximum, median, and minimum energy consumption, respectively. Fig. 3 shows, there is 77% difference between the *random\_max* and the *random\_med* and 90% between the *random\_med* and *random\_min* cases. As we can see, the random mapping does not necessarily find the optimal solution, but it still manages to save almost 90% when compared to the median and minimum solutions. The energy-saving rises when the number of randomly generated mapping increases.

Table 1: The features of Eyeriss spatial DNN accelerator (SPA) and the shapes of the fifth layer of VGG02 convolution tensors (CT).

SPA architecture		CT shapes	
SPA	Eyeriss	CT	Layer 5 VGG_02
On-chip storage levels	2	C	128
DRAM(width)	64	M	256
$L_1$ (depth,width)	(16384, 64)	N	1
$L_0$ (depth,width)	(16,16)	P	56
PE array	(12,14)	Q	56
		R	3
		S	3

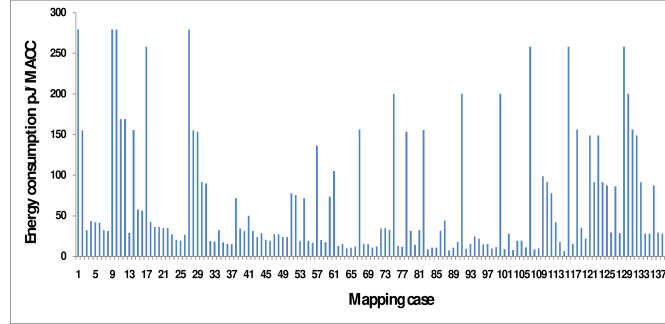


Figure 3: Energy consumption of random mapping.

Based on CT shapes and levels of the memory hierarchy in Eyeriss, the map-space is  $(n!)^m$ , while  $n$  is the number of loop-nests that can be swapped randomly, and  $m$  is the number of storage levels. Since the mapping space of the fifth layer of *VGG02* with six nested loop on Eyeriss with three storage levels is equal to  $O(10^8) = (6!)^3$ . As a result, calculating the energy and latency of all mapping cases with searching by exhaustive brute-force methods will take a long time. In this example, the structure of an accelerator is already specified, but if we need to decide on the number of processing elements, we will have  $O(10^9) = 64^2 \times 224^2 \times 3^2$  design cases for the second layer of *VGG16* ( $K = 64, C = 64, Y = 224, X = 224, R = 3, S = 3$ ).

Thus, the entire design space, including the accelerator configuration and mapping-space, is equal to  $O(10^{17}) = 64^2 \times 224^2 \times 3^2 \times 6!^3$ .

As we can see, this design space is hard to enumerate, so a low-complex mapping algorithm is needed. The following section introduces the *LOCAL* algorithm that finds the nearly optimal energy-efficient solution  $2 \times -38 \times$  faster than other dataflow mechanisms.

## 4 Problem formulation

Given the convolution tensor characteristics and the structure of the spatial DNN accelerator, our objective is to map the convolution tensors (CT) to the spatial DNN accelerator (SPA) such that the energy consumption is minimized under a minimum complex mapping algorithm. Of note, maximizing the number of inferences per second causes maximizing the PE utilization and minimizing energy consumption [23]. More formally:

**Given:** A convolution tensors (CT), and a spatial DNN accelerator (SPA).

**Find:** a mapping function that maps convolution tensors to a spatial DNN accelerator to minimize:

$$\min \{energy\} \quad (23)$$

and maximizing:

$$\max \{PE \text{ utilization}\} \quad (24)$$

such that:

$$utilization \text{ of } PEs = \frac{\text{number of active } PEs}{\text{number of } PEs} \quad (25)$$

---

**Algorithm1:** Tensor to memory assignments  $ct_i$  to  $s_{i(j,k)}$

---

Inputs: All  $ct_i \in CT$  and all  $s_{i(j,k)} \in S$   
 Size of  $(s_{i,j,k})$   
 PE dimension ( $PE_{m,n}$ )  
 Outputs:  $ct_i \rightarrow s_{i(j,k)}$  (Assignment & Permutation)  
 $ct_i \rightarrow PE_{i \rightarrow j(x)y}$  (Parallelization)

```

1: Parallelization:
2:  $\forall s_{i=1,j,k}$ :
3:   if ( $k=0$ ) //NVDLAstyle
4:     assign: parallel_for C in  $Rang(m)$  spatial x dimension to  $s_{i=1}$ 
5:     assign: parallel_for M in  $Rang(n)$  spatial y dimension to  $s_{i=1}$ 
6:   else //Eyeriss Style
7:     assign: parallel_for in Q in  $Rang(m)$  spatial x dimension to  $s_{i=1}$ 
8:     assign: parallel_for in S in  $Rang(n)$  spatial y dimension to  $s_{i=1}$ 
9:   end_if
10: Assignment:
11:  $\forall$  all Unassigned tensors:  $U_{ct_i}$ 
12: for all remaining  $U_{ct_i}$ :
13:   for ( $i=0 \rightarrow i \leq n$ )
14:     Assign  $U_{ct_i}$  to  $s_i$  // Assigning with priority from  $s_{i=0}$  to  $s_{i=n}$ 
15:   end_for
16: end_for
17: Scheduling:
18: for all Assigned tensors:  $A_{ct_i}$ 
19:   sort high to low range  $ct_i$ 
20:    $\forall s_i$  for ( $i=0 \rightarrow i \leq \max$  memory level)
21:     do permutation to allocate higher range tensor to lower  $s_i$ 
22:   end_for

```

---

Figure 4: The pseudo code of *LOCAL* algorithm.

Accordingly, high PE utilization is achieved by keeping the maximum number of PEs active, and this is achieved by proper data assignment and operation scheduling.

## 5 LOCAL mapping algorithm

The main feature of the LOCAL mapping algorithm is low complexity, which provides an appropriate mapping in one pass and a short amount of time. The mapping algorithms that were previously proposed [18][19] were iterative and had long execution times. The main idea behind the LOCAL algorithm is to achieve maximum parallelism by performing spatial mapping of effective tensors, because high parallelism leads to increased PE utilization, as presented in Eq.(25). As Fig. 4 shows, the LOCAL algorithm takes convolution tensors ( $ct_i \in CT$ ), storage elements ( $s_{i,j,k} \in S$ ), and dimensions of PEs as input and gives the final mapping as assigned tensors to storage elements with specified ranges, schedules, and parallelized features.

According to Fig. 4, the LOCAL mapping algorithm consists of three main steps, *parallelization*, *assignment*, and *scheduling*.

Parallelization is the first step due to its importance. In this step, the parallelized tensors are considered based on the type of accelerator. According to Fig. 2, Eq. (14-16), the main difference between Eyeriss and NVDLA style is the number of storage blocks at the level  $i = 1$ , and also their connection to an array of PEs. While the type of accelerator is NVDLA-style,  $C$  and  $M$  as effective shapes are mapped spatially on the  $x$  and  $y$  dimensions of PE-array, respectively (lines 3-5), which causes increasing PE utilization as well as energy efficiency (Fig. 5). Likewise, the two most effective shapes in the Eyeriss-style,  $Q$  and  $S$ , are mapped in parallel to the  $x$  and  $y$  dimensions of PE-array, respectively (lines 7 and 8) (Fig. 5).

The next step, called an assignment, assigns the rest of the unassigned tensors to memory elements with priority from the lowest to the highest level (lines 11-16).

The final phase is scheduling, which permutes assigned tensors to give lower-level memory elements larger range due to the lower energy cost at lower levels of memory than at higher levels (Lines 18-22). After running the LOCAL algorithm, the assigned tensors with their range, parallelization, and loop-scheduling are determined to be the algorithm's output.

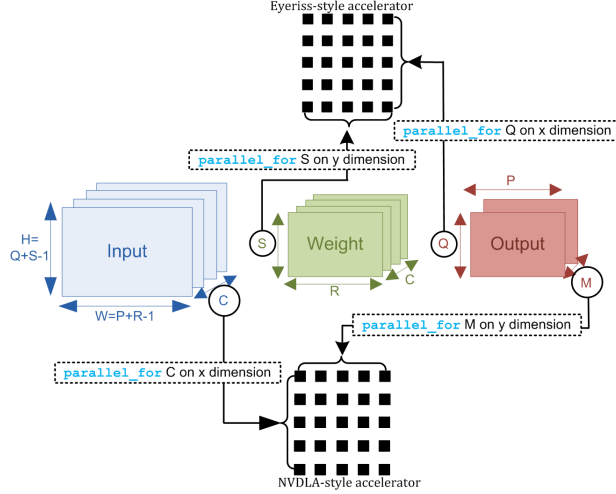


Figure 5: Tensor spatial mapping (parallel\_for) in Eyeriss and NVDLA-like accelerators.

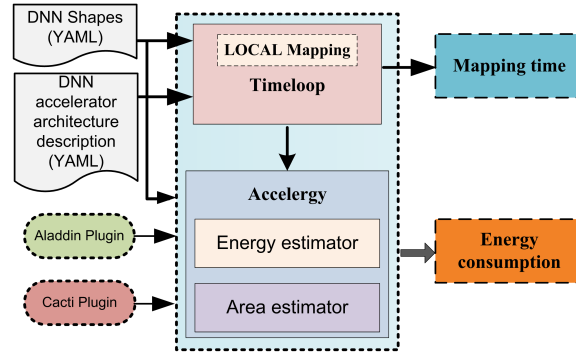


Figure 6: Simulation framework.

## 6 Evaluation

### 6.1 Method

To simulate the LOCAL algorithm and compare it with other dataflow mechanisms, we added the LOCAL mapping to the source code of the Timeloop-Accelergy framework [24][25][26]. Fig. 6 shows the simulation framework.

### 6.2 Simulation Workloads

To achieve a fair simulation, we categorized the workloads based on the magnitude of four main parameters of the convolution layers, including  $C$ ,  $M$ ,  $P$ , and  $Q$ . This category, with more details about the number of MAC operations, is shown in Table 2. We applied our proposed mapping to the Eyeriss, NVDLA, and ShiDianNao accelerators and compared it with the row, weight, and output stationary, respectively. Comparisons are made based on two main parameters: energy consumption and mapping time. Mapping time is based on seconds and is equal to the duration of time it takes to find the proper map. Table 3 shows the mapping times for Eyeriss, NVDLA, and ShiDianNao with LOCAL mapping and their dataflows. The calculation time of row, weight, and output stationary are extracted from the Timeloop-Accelergy framework by defining data-reuse constraints. Furthermore, the mapping time of the LOCAL algorithm is evaluated based on employing the LOCAL mapping function inside the Timeloop. It should be noted that the different mapping times of the LOCAL algorithm in Table 3 are due to the variation of DNN layer shapes.

As can be seen from Tables 3, LOCAL reaches  $34\times$ ,  $38\times$ , and  $49\times$ , faster calculation time than row, output, and weight stationaries. Since in those dataflows, we still need many comparisons to select the appropriate case despite the definition of numerous constraints. It is noteworthy that we have several cases in each stationary method that differ due to parallel for-loops. For this reason, we need many comparisons in choosing the proper case.



Table 2: Workload categories

Category	Workload	Number of MAC operations
High C value	22 <sup>nd</sup> conv layer of Resnet50	51380224
	23 <sup>rd</sup> conv layer of SqueezeNet	5537792
	9 <sup>th</sup> conv layer of VGG16	1849688064
High M value	25 <sup>th</sup> conv layer of SqueezeNet	24920064
	24 <sup>th</sup> conv layer of ResNet50	51380224
	8 <sup>th</sup> conv layer of VGG16	924844032
High P and Q values	1 <sup>st</sup> conv layer of SqueezeNet	708083712
	1 <sup>st</sup> conv layer of ResNet50	472055808
	1 <sup>st</sup> conv layer of VGG16	86704128

Table 3: The mapping time of Eyeriss, NVDLA, and ShiDianNao with LOCAL mapping and their dataflows.

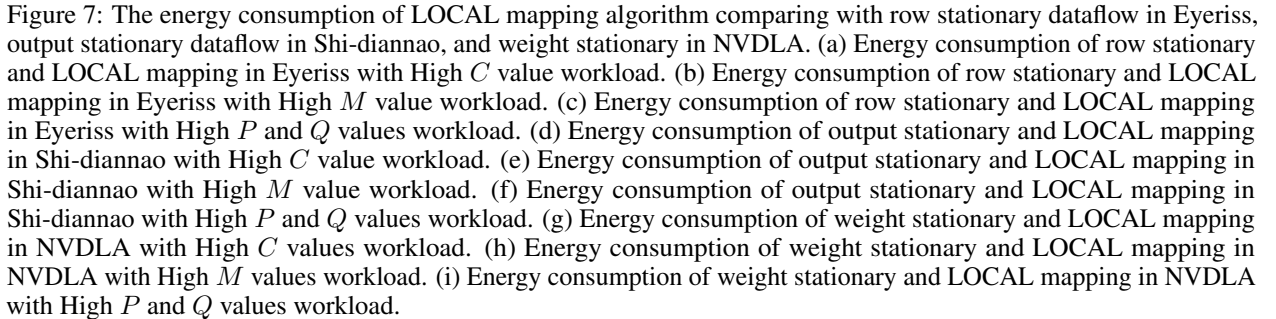
Workload	Convolution	Mapping mechanism	Mapping time (sec)	Mapping mechanism	Mapping time (sec)	Mapping mechanism	Mapping time (sec)
<b>High C value</b>	Resnet50: Conv 22	RS	87	OS	576	WS	127
		LOCAL	16.2	LOCAL	15	LOCAL	6
	VGG16: Conv 9	RS	170	OS	137	WS	68
		LOCAL	10	LOCAL	15	LOCAL	9
	SqueezeNet: Conv 23	RS	17	OS	125	WS	21
		LOCAL	16	LOCAL	67	LOCAL	18
<b>High M value</b>	SqueezeNet: Conv 25	RS	<b>230</b>	OS	126	WS	996
		LOCAL	<b>6.6</b>	LOCAL	16	LOCAL	31
	Resnet50: Conv 24	RS	74	OS	116	WS	42
		LOCAL	22	LOCAL	28	LOCAL	12
	VGG16: Conv 8	RS	351	OS	98	WS	411
		LOCAL	12	LOCAL	32	LOCAL	24
<b>High P and Q values</b>	SqueezeNet: Conv1	RS	60	OS	20	WS	<b>2238</b>
		LOCAL	5.1	LOCAL	7	LOCAL	45
	Resnet50: Conv 1	RS	90	OS	60	WS	140
		LOCAL	6	LOCAL	13	LOCAL	23
	VGG16: Conv 1	RS	81	OS	24	WS	113
		LOCAL	6.6	LOCAL	6	LOCAL	17

Fig. 7 shows the energy consumption of previously proposed dataflows including, row, output, and weight stationary for Eyeriss, ShiDianNao, and NVDLA accelerators. As expected, a large portion of the energy consumption is related to DRAM, and we can also conclude that if the amount of data movement between memory elements and the array of processing elements is reduced, we can achieve higher energy efficiency. Another conclusion that can be drawn from the results is that the LOCAL algorithm has achieved acceptable results in terms of energy consumption in a short processing time compared to other dataflows.

## 7 Related Works

DNN accelerators, like any other processing hardware, have two essential design points: components and data. Components are primary hardware resources, and data is the input of an accelerator. The focus of this paper is on the data side, more precisely, the data mapping design point. A basic concept of dataflow and mapping method is reusing data to reduce data movement between memory to PE-array, and PE to PE. Based on the category of [14] the primary dataflow are, input [11], output [15], [27], [28], weight [4], [29], [30], [31], [32], row stationary [2] and no local reuse [33].

Mapping strategy extended the idea of dataflow by reusing multiple parameters with specified ranges based on the shape of DNN and hardware resources of an accelerator. For instance, we can cite mRNA [16] for MAERI [13] and general methods including Marvel [17], dMazeRunner [34] and interstellar [35] that improves energy consumption.



Heuristic algorithms based on evolutionary [19] or learning-based [18] methods have been proposed to improve energy consumption.

## 8 Conclusion

10

## 9 Acknowledgment

This work was supported, in part, by Science Foundation Ireland under grant No. 13/RC/2094\_P2, co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero<sup>2</sup> (Science Foundation Ireland Research Centre for Software), and, in part, this project has received funding from the European Union’s Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 754489.

## References

- [1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.
- [2] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 44(3):367–379, 2016.
- [3] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th annual international symposium on computer architecture*, pages 1–12, 2017.
- [4] NVDLA deep learning accelerator. URL <https://nvdla.org/>.
- [5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- [6] Alexandros Karatzoglou and Balázs Hidasi. Deep learning for recommender systems. In *Proceedings of the eleventh ACM conference on recommender systems*, pages 396–397, 2017.
- [7] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [8] Giuseppe Ascia, Vincenzo Catania, Salvatore Monteleone, Maurizio Palesi, Davide Patti, and John Jose. Networks-on-chip based deep neural networks accelerators for iot edge devices. In *2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, pages 227–234. IEEE, 2019.
- [9] Kun-Chih Jimmy Chen, Masoumeh Ebrahimi, Ting-Yi Wang, Yuch-Chi Yang, and Yuan-Hao Liao. A noc-based simulator for design and evaluation of deep neural networks. *Microprocessors and Microsystems*, 77:103145, 2020.
- [10] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. 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): 292–308, 2019.
- [11] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *ACM SIGARCH Computer Architecture News*, 42(1):269–284, 2014.
- [12] Yakun Sophia Shao, Jason Clemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–27, 2019.
- [13] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. Maeri: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices*, 53(2):461–475, 2018.
- [14] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Using dataflow to optimize energy efficiency of deep neural network accelerators. *IEEE Micro*, 37(3):12–21, 2017.
- [15] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 92–104, 2015.
- [16] Zhongyuan Zhao, Hyoukjun Kwon, Sachit Kuhar, Weiguang Sheng, Zhigang Mao, and Tushar Krishna. mrna: Enabling efficient mapping space exploration for a reconfiguration neural accelerator. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 282–292. IEEE, 2019.

<sup>2</sup><https://lero.ie/>

- [17] Prasanth Chatarasi, Hyoukjun Kwon, Natesh Raina, Saurabh Malik, Vaisakh Haridas, Tushar Krishna, and Vivek Sarkar. Marvel: A decoupled model-driven approach for efficiently mapping convolutions on spatial dnn accelerators. *arXiv preprint arXiv:2002.07752*, 2020.
- [18] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. Confucius: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 622–636. IEEE, 2020.
- [19] Sheng-Chun Kao and Tushar Krishna. Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.
- [20] Seyedeh Yasaman Hosseini Mirmahaleh, Midia Reshadi, Hesam Shabani, Xiaochen Guo, and Nader Bagherzadeh. Flow mapping and data distribution on mesh-based deep learning accelerator. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, pages 1–8, 2019.
- [21] Seyedeh Yasaman Hosseini Mirmahaleh, Midia Reshadi, and Nader Bagherzadeh. Flow mapping on mesh-based deep learning accelerator. *Journal of Parallel and Distributed Computing*, 144:80–97, 2020.
- [22] Baharealsadat Parchamdar and Midia Reshadi. Data flow mapping onto dnn accelerator considering hardware cost. In *2020 IEEE 14th Dallas Circuits and Systems Conference (DCAS)*, pages 1–5. IEEE, 2020.
- [23] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. How to evaluate deep neural network processors: Tops/w (alone) considered harmful. *IEEE Solid-State Circuits Magazine*, 12(3):28–41, 2020.
- [24] Yannan Nellie Wu, Joel S Emer, and Vivienne Sze. Accelergy: An architecture-level energy estimation methodology for accelerator designs. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.
- [25] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Bruce Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A systematic approach to dnn accelerator evaluation. In *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 304–315. IEEE, 2019.
- [26] Local mapping. URL [https://github.com/midiareshadi/LOCAL\\_Mapping](https://github.com/midiareshadi/LOCAL_Mapping).
- [27] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International conference on machine learning*, pages 1737–1746. PMLR, 2015.
- [28] Maurice Peemen, Arnaud AA Setio, Bart Mesman, and Henk Corporaal. Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st international conference on computer design (ICCD)*, pages 13–19. IEEE, 2013.
- [29] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. Origami: A convolutional network accelerator. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, pages 199–204, 2015.
- [30] Srimat Chakradhar, Murugan Sankaradas, Venkata Jakkula, and Srihari Cadambi. A dynamically configurable coprocessor for convolutional neural networks. In *Proceedings of the 37th annual international symposium on Computer architecture*, pages 247–257, 2010.
- [31] Clément Farabet, Berin Martini, Benoit Corda, Polina Akselrod, Eugenio Culurciello, and Yann LeCun. Neuflo: A runtime reconfigurable dataflow processor for vision. In *Cvpr 2011 Workshops*, pages 109–116. IEEE, 2011.
- [32] Seongwook Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. 4.6 a1.93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications. In *2015 IEEE International Solid-State Circuits Conference (ISSCC) Digest of Technical Papers*, pages 1–3. IEEE, 2015.
- [33] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 161–170, 2015.
- [34] Shail Dave, Aviral Shrivastava, Youngbin Kim, Sasikanth Avancha, and Kyoungwoo Lee. Dmazerunner: Optimizing convolutions on dataflow accelerators. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1544–1548. IEEE, 2020.
- [35] Xuan Yang, Mingyu Gao, Qiaoyi Liu, Jeff Setter, Jing Pu, Ankita Nayak, Steven Bell, Kaidi Cao, Heonjae Ha, Priyanka Raina, et al. Interstellar: Using halide’s scheduling language to analyze dnn accelerators. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 369–383, 2020.

- [36] Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *IEEE micro*, 40(3):20–29, 2020.