AERO: Design Space Exploration Framework for resource-constrained CNN mapping on Tile-based Accelerators

Simei Yang, Debjyoti Bhattacharjee, Vinay B. Y. Kumar, Saikat Chatterjee, Sayandip De, Peter Debacker, Diederik Verkest, Arindam Mallik and Francky Catthoor, *Fellow, IEEE*

***Abstract*—Analog In-Memory Compute (AIMC) arrays can store weights and perform matrix-vector multiplication operations for Deep Convolutional Neural Networks (CNNs). A number of recent efforts have integrated AIMC arrays into hybrid digital-analog accelerators in a multi-layer parallel manner to achieve energy efficiency and high throughput. Multi-layer parallelism on large-scale tile-based architectures need efficient mapping support at the processing element (PE)-level (*e.g.*, digital or analog processing elements) and tile-level, which requires fast and accurate design space exploration (DSE) support.**

In this paper, a DSE framework, AERO, is presented to characterize a CNN inference workload executing on a hybrid tile-based architecture that supports multi-layer parallelism. Three characteristics can be seen in our DSE framework: (1) It presents a hierarchical Tile/PE-level mapping exploration strategy including inter-layer interaction, and allowing layer fusion/splitting configurations for PE-level mapping optimization. (2) It unlocks different power-area-performance exploration points under both sufficient and limited resource constraints, while limited resource case is not considered in prior works of multi-layer parallel architectures. The impact of weight loading and weight stationary mapping are accessed for better insights into hybrid tile-based architectures. (3) It incorporates a detailed PPA (Performance, Power, Area) model that supports a broad range of hybrid digital and analog units in a tile. Experimental case-studies are performed for realistic and relevant benchmarks such as MLP, CNNs (Lenet-5, Resnet-18,-34,-50,-101).

***Index Terms*—Design space exploration, hybrid digital-analog accelerator, multi-layer parallelism, inference, AIMC, resource constrained**

I. Introduction

DEEP learning applications, such as deep convolutional neural networks (CNNs) have achieved remarkable breakthroughs in various application domains (*e.g.*, image classification and speech recognition). These workloads involve intensive matrix-vector multiplication (MVM) operations and huge data communication (*e.g.*, activation and weights of CNNs). As CNNs grow deeper and deeper, the ever-increasing computing and communication requirements have led to the need for custom accelerator designs. Dedicated accelerators [1], [2] have been designed to speed up computation using a high degree of parallelism, and optimzed data storage and data movement, in order to improve energy efficiency.

In recent years, analog in memory compute (AIMC) arrays have attracted much attention in the accelerator design [3]. The AIMC array stores the CNN weights in SRAM (or NVM-resistive) array to achieve high storage density. In addition, an AIMC array can effectively execute multiple/many MVM operations simultaneously, thereby achieving low energy consumption and high throughput [4]. Prior works [5]–[7] integrate AIMC with digital components to create a hybrid digital-analog accelerator to support the execution of full CNN workloads. Digital components are introduced to deal with some non-MVM operations (*e.g.*, Normalization, Pooling) that are not easily adopted in AIMC arrays.

The state-of-the-art accelerator architectures (can be digital or hybrid digital-analog) can be broadly classified into two ways— *single-layer parallel* and *multi-layer parallel*. First, single-layer parallel architectures (*e.g.*, digital [1], [2]) execute a single CNN layer on the architecture at any given point of time, parallelising and speeding up execution of operations within the layer. Execution of a layer is not started until the whole output of the preceding layer(s) has been computed. In such architectures, input and output activations are typically stored in the same memory hierarchies (*e.g.*, DRAM, buffer in Eyeriss [2]). Many DSE frameworks (*e.g.*, Timeloop [8], Interstellar [9], Zigzag [10]) have merged to explore loop transformation at memory hierarchies for high data reuse to optimize energy efficiency.

Second, multi-layer parallel architectures (*e.g.*, ISAAC [5] and PUMA [7]) are capable of processing multiple CNN layers simultaneously, allowing multi-layer pipelines to maximize the throughput of a full CNN workload. Such an architecture is often organized into multiple tiles, where each tile can be composed of multiple processing elements (PE) containing AIMC arrays and digital components. Typically, all the tile resources are partitioned across different CNN layers. A given CNN layer (with MVM operations) is processed by some AIMC arrays, while the outputs of the layer are fetched by other AIMC arrays to process the next CNN layer and so on. This helps in reduction of intermediate result storage space as well as speeding up execution [6]. Unlike DSE frameworks for single-layer parallel architectures (*i.e.*, considering loop mapping at memory hierarchies), the existing DSE frameworks/methodologies for multi-layer parallel architectures (*e.g.*, in [5]–[7], [11]) and they focus more on mapping at AIMC-level and Tile/PE-level.

In this paper, a systematic design space exploration (DSE) framework, AERO, is presented to allow early and fast evaluations of a CNN inference workload executing on a hybrid multi-layer parallel tile-based architecture that integrates both multiple AIMC arrays and digital components in each tile. The AERO framework presents a complete mapping flow, including *virtual mapping* and *physical mapping* (hierarchical mapping exploration) at tile-level, PE-level and AIMC-level. Two key novelties can be seen in our DSE framework.

- The proposed framework supports hierarchical mapping in multi-layer parallel architecture under both sufficient and limited resource constraints (available tiles/PEs). Existing works consider sufficient tiles/PEs to execute all layers of a full CNN simultaneously, assuming weights (of all layers) are pre-loaded into sufficient AIMCs. AERO unlocks different power-area-performance exploration points under limited resources, which is not possible in prior efforts. Particularly, the impact of weight loading and weight stationary are assessed to gain a better insight into hybrid tile-based architectures.
- The proposed framework incorporates detailed PPA (Performance, Power, Area) models, which support a broad range of hybrid digital and analog units within a platform tile, as well as instruction memory and weight loading overhead. Most existing DSE frameworks do not take into account instruction memory/weight loading overhead (having a big bottleneck) or they consider quite high-level estimations (*e.g.*, for MAC computation and memory access overhead as in Timeloop [8], Interstellar [9], ZigZag [10]).

Moreover, experimental case-studies for MLP, (deep) CNNs (Lenet-5, Resnet-18,-34,-50,-101) offer insights into the impacts of design parameters, *e.g.*, (1) weight loading and weight stationary, (2) external memory access bandwidth of weights, (3) sufficient / limited resource constraints, (4) AIMC dimensions and (5) AIMC memory cell technology choices, on hybrid tile-based systems.

The rest of the paper is organized as follows. Section II compares the existing DSE frameworks for accelerators and highlights the novelties of our work. Section III introduces a hybrid digital-analog tile-based accelerator template as an experimental benchmark in our DSE framework. Section IV describes the mapping methodology and PPA estimation approach of the proposed DSE framework (first contribution). Section V discusses the insights derived from the experimental case-studies (second contribution). Section VI discusses the advantages and limitations of the current AERO framework. Section VII concludes our work.

II. Related Work

Table I compares the state-of-the-art DSE frameworks (or methodologies) related to *single-layer parallel* and *multi-layer parallel* CNN architectures, in terms of *mapping approaches* (*i.e.*, at AIMC level, Tile/PE-level, at memory hierarchies) and *modeling supports* (*e.g.*, on PPA models, resource constraint).

On one hand, multiple systematic DSE frameworks, such as Timeloop [8]+Accelergy [12], Interstellar [9], Zigzag [10], focus on convolution (Conv) or Fully-connected (FC) layers executing on *single-layer parallel* architectures. These frameworks are primarily designed towards exploration of digital architectural templates (*e.g.*, Eyeriss [2]) and corresponding mapping opportunities. The digital architectural template consists of a digital processing element (PE) array, along with the required memory hierarchies. A variant of Zigzag [4] considers an analog in-memory compute (AIMC) array alongside the required memory hierarchies. These DSE frameworks adopt loop transformation approaches into the mapping flow to represent and analyze the design space of Conv/FC layers mapped at PE/AIMC arrays (see ① column in Table I) and memory hierarchies (see ③ column).

Particularly for the mapping at memory hierarchies (see ③ column), *single-layer parallel* architectures typically consider input activations coming from DRAM to buffers (*e.g.*, L1 and/or L2 buffer) and PE/AIMC array, while the output activations are stored in the same memory hierarchies. The mapping flow is targeted to find energy-efficient solutions that increase the data reuse within the PE/AIMC array and also at different levels of the memory hierarchy. It is worth mentioning that these DSE frameworks consider high-level PPA models (see ④ column), focusing on MAC computation and memory access overhead.

On the other hand, there has been some works [5]–[7], [11], [13] consider *multi-layer parallel* architecture to support the execution of a full CNN. Typically, the *multi-layer* parallel architectures are organized in a tiled manner – one or more compute arrays (either digital or analog) are grouped together in the form of a tile and tiles communicate using network-on-chip (NoC). These architectures take considerably larger area compared to the *single-layer* parallel architectures and are suitable for high performance scenarios.

For AIMC-level mapping (*i.e.*, mapping weights onto AIMC arrays, see ① column), no exploration is performed in the works of [5]–[7], [11], [13]. These existing works are based on the conventional weight mapping method, aiming to further increase the utilization and data reuse of AIMC arrays to improve computational parallelism and energy efficiency. In the conventional approach, a 3D Conv kernel (*e.g.*, kx, ky, cin as in Fig. 3) is transformed across AIMC rows, and multiple 3D kernels (*e.g.*, cout as in Fig. 3) can be mapped across AIMC columns. Our work also adopts the conventional weight mapping in AIMCs for high utilization. More details about the assumption can be seen in Section IV-A.

For tile/PE-level mapping (see ② column), the objective is to map *nodes* (*e.g.*, layers or operations) to tiles and PEs. The works of [5], [6], [11] perform the tile-level mapping manually without exploration. PUMA [7] explores the optimized tile/PE-level by a hierarchical partitioning strategy in *bottom-up* manner (*i.e.*, from PEs to tiles to accelerator). In contrast, a *top-down* hierarchical partitioning strategy (*i.e.*,from accelerator to clusters, to tiles/PEs) in used in our work, to reduce inter-tile communication traffic within reasonable exploration time. Moreover, our proposed DSE framework supports *node fusion* and *splitting* configurations (at PE-level level) as in TVM compiler[1] [15]. Similar to TVM, splitting factors are configured manually in the current AERO framework. However, fusion and splitting configurations are not supported in PUMA [7], though *splitting* is discussed in [5], [6].

[1]TVM compiler does not directly/currently support *multi-layer parallel* architectures in the published version [14].

TABLE I
COMPARISON OF RELATED WORK ON CNN ACCELERATORS

Reference	**Architecture & Workloads**	**Mapping approaches**			**④ Modeling supports**
		① At PE/AIMC-level	**②At Tile/PE-level**	**③At memory hierarchies**	
Timeloop [8] + Accelergy [12]	-Single-layer parallel architecture - Conv and FC	- Exploration: spatial mapping in AIMC/PE array - No tile-level mapping		- Exploration: loop transformation in L1, L2, DRAM	- PPA: high-level, considering MAC computation and memory access overhead
Interstellar [9]					
Zigzag [4], [10]					
ISAAC [5]	-Multi-layer parallel architecture - Full neural network	- Conventional mapping in AIMC arrays - Exploration: No	- Exploration: No (manually) - NF: No	- Analyze buffer requirement between two layers	- PPA: detailed, excluding Instr Mem & WL overheads - RC: sufficient
PUMA [7], [13]			- Exploration: HPA (bottom-up)	- Consider register allocation at compiler level	- Instr: compiled ISA - PPA: detailed, excluding WL overhead - RC: sufficient
[6]		- Improve AIMC data reuse w.r.t. conventional approach - Exploration: No	- Exploration: No - NS: Yes - NF: No	- Discuss buffer requirement between two layers	- PPA: detailed, exlucing Instr Mem & WL overheads - RC: sufficient
[11]		- Improve AIMC utilization w.r.t. conventional approach - Exploration: No	- Exploration: No - NS: No - NF: No	- No exploration - Consider DP of activations in input Mem	- PPA: high-level, excluding WL overhead - RC: sufficient
This work		- Conventional mapping in AIMC array, aligned to [4] - Exploration: No	- Exploration: HPA (top-down) - NS: Yes - NF: Yes	- Assume small buffer between two layers as in [5] - Assume ideal DP of activations in input Mem	- Instr: approximate ISA - PPA: detailed, considering Instr Mem & WL overheads - RC: sufficient & limited
Note: NS: node splitting, **NF**: node fusion, **HPA**: Hierarchical partitioning algorithm, **Instr**: instructions, **Mem**: memory, **WL**: weight loading, **RC**: resource constraint, **DP**: data placement. * The gray cells highlight the main difference between the closest related work (PUMA) and our work.					

For mapping at memory hierarchies (*i.e.*, loop transformation, see ③ column), the existing works of [5]–[7], [11] do not perform exploration as in Timeloop [8] and Zigzag frameworks [10] (for *single-layer parallel* architectures). As previously discussed, *single-layer parallel* architectures store input and output activations of a layer in the same memory hierarchies. Differently, *multi-layer parallel* architectures typically store the output activations of one layer in the buffer of a neighboring PE as input activations for the next layer. The next layer can start computation without having to obtain all the outputs of previous layer[2]. This can be further understood as implicit assumptions in existing works ([5]–[7], [11]), that is, all loops related to output activations are ideally mapped to the buffer of a neighboring PE and the next layer can consume data before the buffer is full. This assumption still holds in our work. Besides, the AERO framework currently assume ideal data placement in input activation buffer (*e.g.*, SRAM), where the required data for generating an output is continuously available. More investigations on mapping *loops* at memory hierarchies and orchestrating data in input activation buffer (such as in [11]), will be considered in future work.

Our work is further compared to existing works in two more aspects (see ④ column). (1) While PUMA is a compiler which generates compiled ISA (instruction set), our AERO framework can be regarded as a preliminary step before a real compiler. AERO generates *approximate* ISA, assuming ideal data placement as previously discussed. The approximate instructions are further used to derive information (*e.g.*, action counts of different hardware components, ISA execution cycles) for detailed PPA (*e.g.*, latency, energy) models. Most of the existing works exclude instruction memory and weight loading overheads (*e.g.*, [5], [6]), or they consider quite high-level estimations (*e.g.*, [8]–[11]). (2) The works of [5]–[7], [11] consider that the number of tiles is always sufficient to execute all neural network layers simultaneously, and weights of all layers are pre-loaded into sufficient AIMCs. This explains why weight loading overhead is excluded in their PPA models. In contrast, AERO considers the cases of *sufficient* and *limited* number of available tiles/PEs, thereby unlocking different energy-area-performance exploration points that are previously not possible. AERO also assesses the impact of weight loading to achieve better insight into hybrid tile-based architectures.

[2]The buffer between two layers can be in small size, which is about $inx \times ky \times cin$, where inx is the number of columns in input feature, ky is the number of rows in kernel, cin is the number of channels in input feature.

III. ARCHITECTURE TEMPLATE

This section introduces a multi-layer Tiled Analog In-Memory Accelerator (TANIA) architecture template, which is used for design space exploration. For reference, Table II lists the parameters in the architecture template and other variables used in rest of this paper. Fig. 1(a) illustrates the top level TANIA architecture template. It comprises of multiple clusters, with two external interfaces – host interface and a weight bank interface. Each cluster consists of multiple tiles, which share a local (*e.g.*, L2) buffer for storing intermediate results if needed. The tiles communicate over NoC for transferring activations.

Each tile consists of multiple PEs, consisting of N_a analog Aicores [16] and N_d digital Vector Functional Units (VFUs), as shown in Fig. 1(b). As introduced in [16], each Aicore has an AIMC array to performs MVM operations and it has additional digital circuits for batch-normalization (BN) and non-linearity (NL) operations. This offers the possibility to immediately process the MVM results through BN or NL

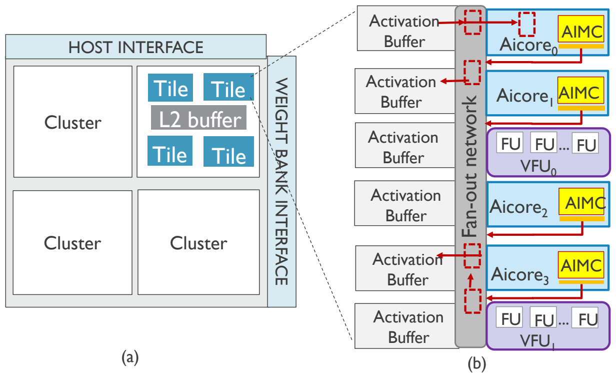


Fig. 1. TANIA architecture template: (a) description of the accelerator-level with multiple clusters; (b) description of the tile-level.

TABLE II
NOTATIONS USED IN THIS PAPER

Architecture template	
$N_{cluster}$	Number of clusters in the TANIA template
N_{tile}	Number of tiles in a cluster
$J = N_{cluster} \times N_{tile}$	Total number of tiles in the TANIA
N_a	Number of analog Aicores in a tile
N_d	Number of digital VFUs in a tile
$N_a + N_d$	Number of activation buffers in a tile
Neural network inference workload	
$Conv$	Convolution
$Gemm$	General Matrix Multiply
$Psum$	Partial Sum
$Bias$	Bias add
Add	Residual add
BN	Batch Normalization
NL	Non-Linearity
Virtual mapping	
$node_a$	A virtual node targeting an analog Core
$node_d$	A virtual node targeting a digital VFU
$node_r$	A virtual node targeting data reshape
I	Total number of virtual nodes
K	Total number of virtual communication edges
Physical mapping	
SR	Sufficient resource constraint
LR	Limited resource constraint
WL	Weight loading mode
WS	Weight stationary mode

operations to reduce the amount of intermediate data. For simplicity, the number of BN and NL is considered to be equal to number of AIMC columns. Notice that Aicore and AIMC are not the focus of this paper. More details about them can be found in [16], [17]. Similar to [7], each VFU is a SIMD unit, which concurrently performs digital operation on multi-channel (*e.g.*, 64) activations for high throughput. Analog Aicores and digital VFUs support different operations, which are summarized in Table III. Each PE has a local activation buffer. A low fan-out network is used for communication between the PEs, which offers a much lower energy cost for data transfer within the tile, compared to inter-tile data transfer (via NoC). The output of a PE is written to the activation buffer of the destination PE either L2 buffer within a cluster.

Each PE has a local instruction memory and program counter. This allow PEs to operate independently, with the flexibility o f m apping l ayers t o a rbitrary P Es. S ince t he cost of inter-tile communication is higher than intra-tile communication, a mapping strategy is required to map layers in a way that reduces inter-tile communication. The mapping framework will be presented in the following section.

TABLE III
SUPPORTED OPERATIONS FOR AICORE AND VFU

Operations	Supported PE types
Conv, Gemm	Aicore
Pooling, Residual Add, Partial Sum (Psum)	VFU
Bias, Batch Normalization (BN), Non Linear (NL)	Aicore or VFU

IV. AERO DESIGN-SPACE-EXPLORATION FRAMEWORK

The overview of the proposed DSE framework, AERO, is shown in Fig. 2. The framework takes a hardware architecture template and a neural network (*e.g.*, CNN) inference workload as input. TANIA architecture template (See Section III) is used as the base hardware architecture in the framework. The inference workload is imported in Open Neural Network Exchange (ONNX) format [18]. ONNX format is introduced to enable interoperabililty among a variety of frameworks [18]. Microsoft NNI [19] is leveraged for performing multiple DSE experiments in parallel locally or on a cluster.

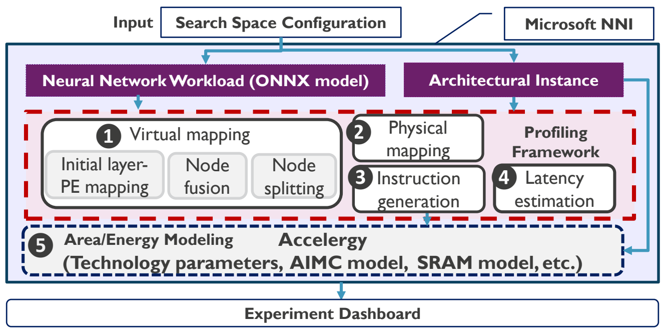


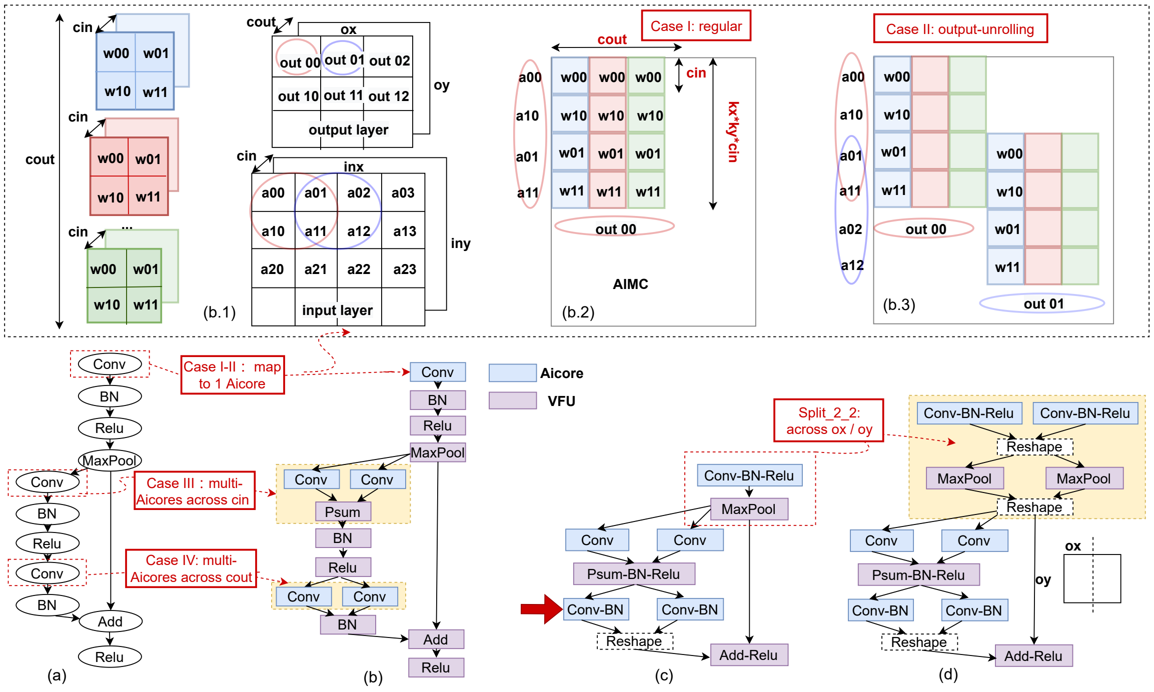
Fig. 2. Overview of AERO DSE framework.

AERO framework encompasses a generic mapping methodology, which consists of virtual mapping (see ❶ in Fig.2) and physical mapping (see ❷). This generic mapping strategy can be applied to inference workloads executed on different accelerator templates (*e.g.*, ISAAC [5], PUMA [7]). Based on mapping solution, preliminary instructions (assuming ideal data placement as discussed in Section II) are generated for each PE (see ❸). Additionally, a latency characterization approach (see ❹) is presented to model the start time and finish time of each pixel computation, considering inter-layer data dependencies. To estimate the area of an architecture template, and the energy cost for a given workload mapped to it, the Accelergy [12] framework is integrated into the AERO DSE flow (see ❺). A number of component-level estimation plugins specific to the architecture (e.g., for AIMC array, NoC, and other building blocks) are added for energy computation. The generated results are displayed in a dashboard.

The following sections will present the details of mapping and PPA modeling. The custom ISA for instruction generation (see ❸) is out of the scope of this paper.

A. Virtual mapping

Virtual mapping (❶ in Fig. 2) aims to map each neural network layer to Aicores or VFUs, with respect to AIMC dimension constraints. Virtual mapping implies that the resource availability (*i.e.*, Aicores and VFUs) of the physical platform is not considered in this stage. Virtual mapping is fulfilled in



Case I: regular
Case II: output-unrolling
Case I-II : map to 1 Aicore
Case III : multi-Aicores across cin
Case IV: multi-Aicores across cout
Split_2_2: across ox / oy
Aicore
VFU
output layer
input layer
AIMC
kx*ky*cin
(a)
(b)
(b.1)
(b.2)
(b.3)
(c)
(d)

Fig. 3. Overview of virtual mapping. (a) Part of application graph of Resnet-18. (b) Virtual initial layer-PE mapping: Case I-II maps a MVM layer to one AIMC array (*i.e.*, one Aicore); Case III-IV map a MVM layer to multiple AIMC arrays by diving a Conv kernel across *cin* and *cout* dimension, respectively. (b.1) Example of a Conv layer with kernel, input and output features.(b.2) In Case I, AIMC mapping without output unrolling. (b.3) In Case II, AIMC mapping with output unrolling. (c) Virtual node fusion. (d) Virtual node splitting.

three steps: *initial virtual mapping*, *virtual-node fusion* and *virtual-node splitting*, as illustrated by the example in Fig. 3.

1) Initial layer-PE virtual mapping: For a neural network graph, initially, each MVM layer (*e.g.*, *Conv* and *Gemm*) is mapped to Aicores, and each digital operation layer is mapped to a VFU (see Fig 3 (a-b)). For Aicores, weights should be appropriately mapped to AIMC arrays (*i.e.*, at AIMC-level). The AIMC-level mapping can be summarized into four cases according to [16]. Case I-II map a MVM layer to one AIMC (*i.e.*, in one Aicore). Case II maps multiple copies of weights to a (large) AIMC array, allowing computing multiple outputs simultaneously (*i.e.*, output unrolling in Fig. 3 (b.3)). Case III-IV map a MVM layer to multiple AIMC arrays by dividing a Conv kernel across input (cin) and/or output channels ($cout$). More details about AIMC mapping can refer to [16].

2) Virtual-node fusion: The mapping of a layer to a Aicore/VFU is denoted (Fig.3 (b)) as a virtual node. The node fusion step fuses the adjacent operations of the same pixel into one PE (see Fig. 3 (c)), aiming to reduce intermediate data transfer at memory hierarchies [20]. Based on the hardware supports shown in Table III, AERO performs node fusion according to the following three operation orders. These orders are usually observed in most of the existing CNN workloads (*e.g.*, MLP, Resnet-18).

- Conv-Bias-BN-NL (fuse into Aicore)
- Psum-Bias-BN-NL (fuse into VFU)
- Add (residual)-Relu (fuse into VFU)

Fusion can be performed even when some operations are skipped in each order. For instance, AERO fuses *Conv-BN* operations for Resnet-18 (see the red arrow in Fig. 3 (c)), while *Bias* and *NL* operations are skipped (in the first order). AERO currently incorporates a first fusion heuristic which will be explored more in-depth in the future work.

3) Virtual-node splitting: AERO performs node splitting to allow different PEs to generate different parts of the output activations (*e.g.*, ox and oy of a Conv layer). This can speed up the the computation of shallow layers (*e.g.*, first couples of layers due to the big size of output dimensions [6]) and consequently reduce the inference latency. In the example of Fig 3 (c), *Split_2_2* refers to using two Aicores and two VFUs for Layer0 and Layer1 respectively. The current AERO framework allows setting splitting factor for every layer manually. A good set of splitting factors can be obtained by adding an explicit search space exploration method, which will be resolved in our future work.

4) Virtual mapping result: A virtual mapping can be characterized by a set of I virtual nodes ($node_i$), and a set of K communication edges ($edge_k$) representing data dependencies among the nodes. The virtual nodes targeting an analog Aicore, a digital VFU and a *Reshape* operation (*e.g.*, for data organization in activation buffer), as shown in Fig. 3 (b-d), are denoted by $node_a$, $node_d$ and $node_r$, respectively.

B. Physical mapping

Physical mapping (❷ in Fig. 2) aims to map virtual nodes to the physical architecture entities: clusters, tiles, Aicores and VFUs. The objective is to optimize inter-tile communication over the NoCs, subject to resource constraints.

1) Problem formulation of physical mapping: **Given:** (1) Virtual mapping result, with I heterogeneous virtual nodes and K communication edges (see Section IV-A4). (2) An architecture template (*e.g.*, TANIA in see Section III).

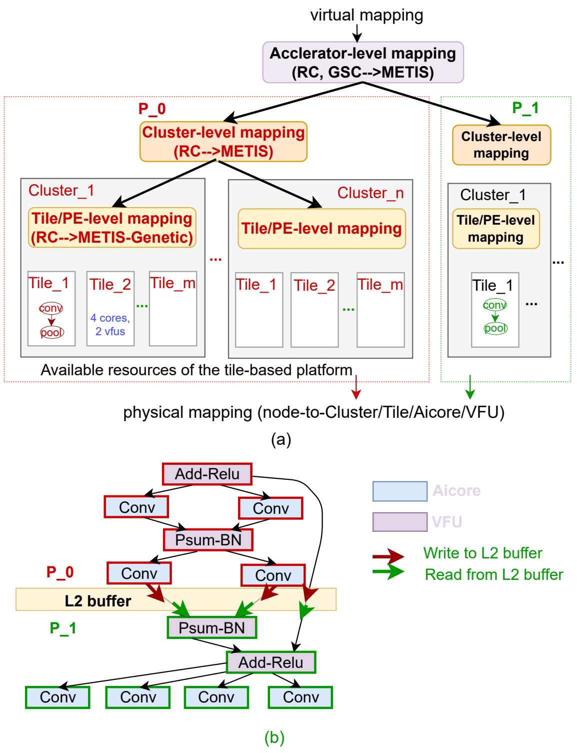

virtual mapping
Acclerator-level mapping (RC, GSC-->METIS)
P_0
P_1
Cluster-level mapping (RC-->METIS)
Cluster-level mapping
Cluster_1
Cluster_n
Tile/PE-level mapping (RC-->METIS-Genetic)
Tile/PE-level mapping
Tile_1
Tile_2
Tile_m
conv
pool
4 cores, 2 vfus
Available resources of the tile-based platform
physical mapping (node-to-Cluster/Tile/Aicore/VFU)
(a)
Add-Relu
Conv
Psum-BN
L2 buffer
Aicore
VFU
Write to L2 buffer
Read from L2 buffer
(b)

Fig. 4. (a) Overview of hierarchical optimization of physical mapping, under resource constraint (RC) and graph sequence constraint (GSC); (b) An example of accelerator-level physical mapping.

Find: An optimized physical mapping, which maps virtual nodes to clusters, tiles, PEs (*i.e.*, Aicores and VFUs). Let the matrix of *binary variables* $[p_{i,j}]_{I\times J}$ represent a physical mapping on tiles (including clusters), where $p_{i,j}$ is defined as:

$$p_{i,j} = \begin{cases} 1, & \text{if } node_i \text{ is assigned to } tile_j. \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

where J refers to the number of all tiles in the architecture (*e.g.*, $J = N_{cluster} \times N_{tile}$ in TANIA). Here, the cluster index is hidden into the tile index ($tile_j$) for clarity of the formula.

Objective: To optimize NoC communication traffic, the inter-tile (including inter-cluster) communication is formulated as in Eq.(2), where $edge_k$ refers to the edge between nodes $node_i$ and $node'_i$. Then the physical mapping objective can be formulated into Eq.(3).

$$Comm_{noc}(p_{i,j}, p_{i',j'}) = \begin{cases} 0, & \text{if } j = j'. \\ edge_k, & \text{otherwise.} \end{cases} \tag{2}$$

$$\min \sum_{i=0}^{I-1} \sum_{j=0}^{J-1} Comm_{noc}(p_{i,j}, p_{i',j'}) \tag{3}$$

Subject to: Resource constraints can be understood by: (1) Each virtual node (except $node_r$) is mapped into one tile (Eq.(4)); (2) The number of $node_a$ and $node_d$ in each tile should respect the tile-level resource availability of Aicores (Eq.(5)) and VFUs (Eq.(6)), respectively.

$$\sum_{j=0}^{J-1} p_{i,j} = 1, \forall i, p_{i,j} \in \{0,1\}, node_i \notin node_r \tag{4}$$

$$\sum_{i=0}^{I-1} p_{i,j} <= N_a, \forall j, node_i \in node_a \tag{5}$$

$$\sum_{i=0}^{I-1} p_{i,j} <= N_d, \forall j, node_i \in node_d \tag{6}$$

Here, the physical mapping at PE-level is not shown for sake of simplification. It is considered that the nodes in each tile can be allocated to any feasible Aicore or VFU. This is based on the assumption that intra-tile communication (*e.g.*, using fan-out networks in TANIA template) is more energy efficient compared to inter-tile communication via NoC.

2) Hierarchical physical mapping strategy: A hierarchical strategy is proposed to achieve optimized physical mapping solution. Mapping on multi/many-cores systems has been proved as a NP-hard problem [21]. The large number of CNN layers (*i.e.*, virtual mapping nodes) and the large scale of hardware resources (*i.e.*, Aicores and VFUs in TANIA) lead to a huge design space. Hierarchical mapping approaches [22], [23] are usually used to resolve the scalability issue. The overview of our hierarchical physical mapping strategy is shown in Fig 4 (a), which consists of three levels.

The proposed hierarchical physical mapping strategy is in *top-down* manner. The accelerator-level mapping divides virtual nodes into multiple partitions. It is particularly proposed for the case where the number of virtual nodes is larger than the number of physical resources (*i.e.*, Aicores and VFUs). The accelerator-level mapping can be further understood as *partitioning / cutting* a neural network to meet resource constraint and graph sequence constraint. It is assumed that the nodes of different partitions execute sequentially at available resources, and the execution of each input node should not be later than its output node (*i.e.*, graph sequence constraint). One example is shown in Fig 4 (b), where virtual nodes are divided into two partitions (*i.e.*, P_0 and P_1). The nodes of P_0 execute on available PEs first (*e.g.*, 4 Aicores and 2 VFUs). After P_0 finishes the execution, the intermediate data between the two partitions are stored into L2 buffer. Then, nodes of P_1 load the intermediate data from L2 buffer to corresponding PEs and start their executions. Afterwards, according to resource constraints, the cluster-level mapping divides each set of partitioned nodes to available clusters, while the tile/PE-level mapping divides each set of clustered nodes to available tiles, as well as PEs.

3) Hierarchical physical mapping strategy implementation: *METIS* [24] graph partitioning program is used at the three hierarchical levels to support efficient explorations. *METIS* is claimed to be one to two orders of magnitude faster than other widely used partitioning algorithms. To further improve solution quality (*i.e.*, reducing inter-tile communication via NoCs, and exploiting the intra-tile communication), on top of *METIS*, a custom genetic algorithm is implemented at tile/PE-level to have local optimization. The combined strategy is denoted by *METIS-Genetic*.

The developed genetic algorithm is based a standard microbial genetic algorithm (MGA) [25]. MGA takes cluster-level physical mapping (*i.e.*, a set of virtual nodes mapped to a cluster) as input to explore tile/PE-level mapping optimization. Firstly, MGA initializes population, which consist of a set of random chromosomes. The METIS solution (at tile/PE-level) is inserted as a basic chromosome. The chromosome of a tile/PE-level mapping solution is decoded as in Fig. 5. The chromosome is a vector of PE's length in a cluster (*i.e.*, $N_a \times N_{tile} + N_d \times N_{tile}$ in TANIA), consisting of two segments for Aicores and VFUs respectively. The chromosome is a permutation of tile index of each PE. In each target cluster, the virtual nodes for Aicores (or VFUs) are sequentially allocated to a tile (with a tile index) according to the Aicore (or VFU) chromosomes. Secondly, the fitness function of a chromosome is defined by $\exp(1/\sum Comm_{noc} + 1))^3$, where $\sum Comm_{noc}$ refers to the inter-tile NoC communication within a cluster (based on Eq.(3)). The smaller the NoC communication, the better the chromosome's fitness is.

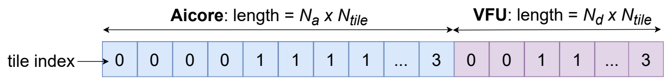

Aicore: length = N_a x N_{tile}
VFU: length = N_d x N_{tile}
tile index
0 0 0 0 1 1 1 1 ... 3 0 0 1 1 ... 3

Fig. 5. The chromosome with two segments for Aicores and VFUs in each cluster. In this example, a cluster has 4 tiles (with tile index from 0 to 3). Each tile has 4 Aicores (4 same tile indexes in blue segment) and 2 VFUs (2 same tile indexes in purple segment).

The proposed hierarchical physical mapping strategy implemented by *METIS-Genetic* is able to exploit up to 82.01% communication data (Bytes) within tiles (*i.e.*, intra-tile), which enables more energy-efficient data transfer than inter-tile data transfer via NoCs (recall Section III). Deep Resnet-34,-50,-101 take $0.01 \sim 0.06$ seconds to obtain METIS solutions, and about $3 \sim 10$ seconds for genetic improvement in each cluster. This indicates that the proposed *METIS-Genetic* is suitable for the architecture template in terms of exploration efficiency and exploration time. The relevant experimental details are not shown due to the limit space of the paper.

C. Latency characterization approach

The proposed latency characterization approach (❹ in Fig. 2) describes start time and finish time of pixel computation, on-chip communication (*i.e.*, intra-tile and inter-cluster/tile communication) and off-chip data transfer behaviors, while taking into account inter-layer data dependencies.

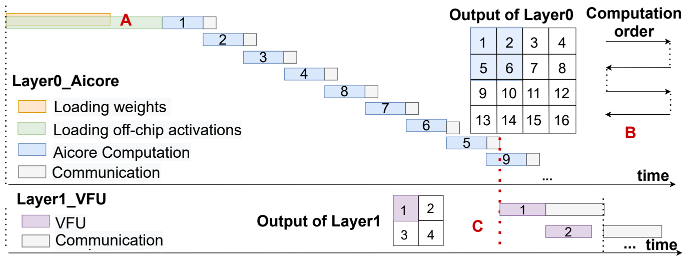


Fig. 6. Latency estimation for Layer0 and Layer1 examples. * For simplicity, the figure characterizes the weight loading (WL) latency in terms of data communication. WL latency caused by AIMC write speed (related to AIMC technology) is discussed in Section V-G.

Fig. 6 illustrates the latency characterization for two example neural network layers, Layer0 (*e.g.*, Conv layer) and Layer1 (*e.g.*, Pooling layer). At the beginning of system execution, weight loading and off-chip activation transferred are performed in parallel through separated NoCs (see A in Fig. 6). Similar to [5], data transfer via NoCs is assumed to be statically routed without any conflicts. In AERO framework, once all off-chip activation (of the first image frame) are loaded from off-chip memory, Layer0 starts to execute according to computation order sequentially (see B in Fig. 6). The pixel computation time is characterized by the average cycles of executing the instructions for one pixel generation, ignoring corner cases (*e.g.*, padding). Computation and inter-tile NoC communication can overlap. NoC communication latency is determined by data size and NoC bandwidth (*i.e.*, NoC communication latency = data size / NoC bandwidth). The generated output pixels are written into the activation buffer (of Aicore/VFU) of the next layer. From the next layer (*e.g.*, Layer1), the computation for an output pixel depends on the maximum latency of the required input activations (*e.g.*, 1,2,5,6 of Layer0, see C in Fig. 6).

Fig. 7 characterizes latency for all layers of a full CNN for sufficient and limited resource constraints. *Multi-layer parallel execution on hybrid tile-based architectures under limited resource constraints is the key novelty in this paper.*

1) Sufficient resources (SR): In this case, the number of physical PEs (Aicore or VFU) is always sufficient for a full neural network (with all virtual nodes). There is no need to use accelerator-level physical mapping strategy (in Section IV-B2) to deal with accelerator resource constraints, but using cluster-level and tile/PE-level strategies. For a set of virtual nodes in Fig. 7 (a), part (b.1) illustrates the data transfer behaviors under SR constraints, while part (b.2) characterizes the corresponding latency behaviors. Initially, the off-chip data transfer, including *weight loading* (WL) for all $node_a$ and *image loading* (IL) for the input node (see IL_0 for $node_a^0$), are performed in parallel. When WL is finished for the first Aicore, it continues for the next Aicore (see A, A' in Fig. 7 (b.2)), and so on. Then, all Aicores and VFUs can start execution with on-chip communication (intra-tile and inter-tile/cluster) depending on data availability. The finish time of the last layer refers to network inference latency (see B). From the second image, the off-chip IL latency (see IL_1 and IL_2) can be hidden in our assumption, and computation and communication can be performed without WL. This mode is typically called *weight stationary* (WS, see C and C'). As a consequence, latency (see B') can be reduced compared to that of the first image.

2) Limited resources (LR): In this case, the number of virtual nodes is larger than the number of physical PEs. It requires accelerator-level physical mapping strategy (see Section IV-B2) to divide the virtual mapping nodes into multiple partitions, followed by cluster-level and tile/PE-level strategies. For the virtual noes in Fig. 7 (a) under LR constraints, part (c.1) describes the data transfer behavior and (c.2) shows the corresponding latency characterization. Due to LR constraints, the virtual nodes are divided into 3 partitions. The off-chip WL is firstly performed for $node_a^{0-3}$ in P_0 (partition index), and IL is performed for $node_a^0$. Then all $node_a$ and $node_v$ in P_0 start computation and on-chip communication (intra-tile, inter-tile/cluster) considering data

[3]1 is added to avoid the case where divisor is 0.

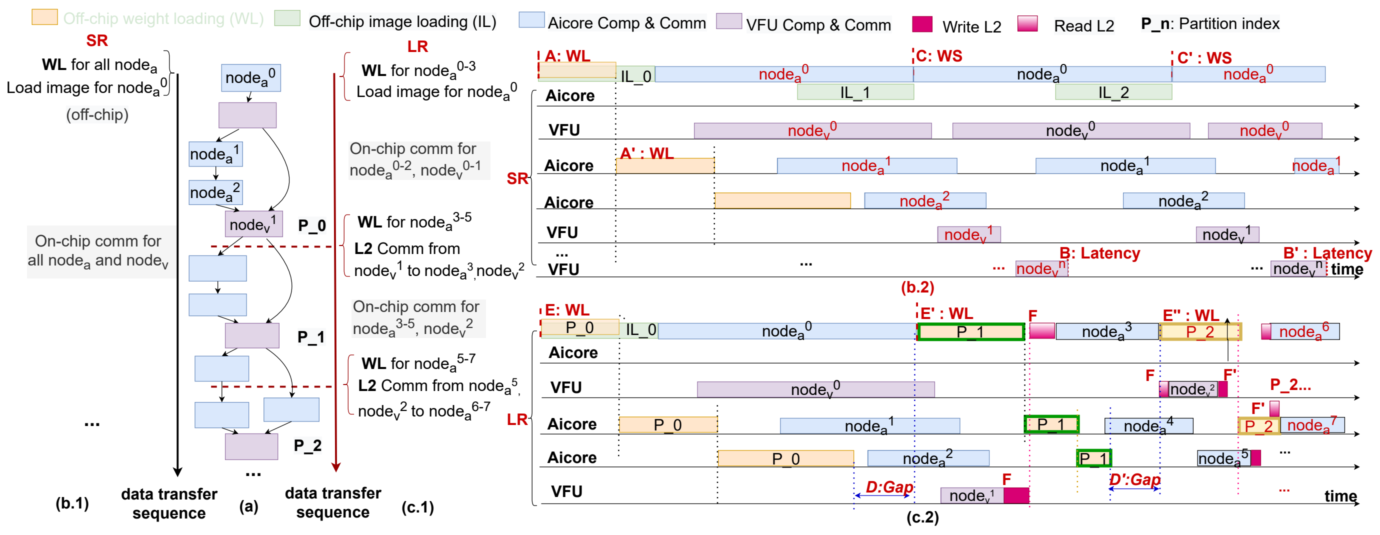
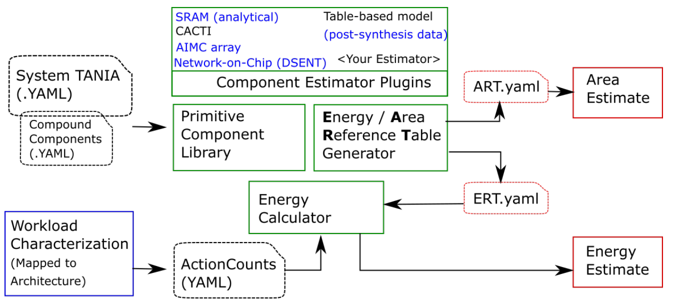


Fig. 7. (a) Example of virtual mapping nodes for a full network. (b.1) Data transfer behaviors of off-chip (WL and IL) and on-chip (intra-tile (via fan-out network), inter-tile/cluster (via NoC), inter-partition (via L2 buffer)) and (b.2) latency characterization, under SR constraint. (c.1) Data transfer behaviors of off-chip and on-chip and (c.2) latency characterization, under LR constraint. **Note:** The *Aicore (or VFU) comm & comm* characterizes the start time of the first pixel and the finish time of the last pixel for an image frame. *Particularly for part (c.2), different nodes mapped to the same Aicore/VFU are displayed in the same row.

availability. When any physical Aicores are released by $node_a$ in P_0, WL can be performed for $node_a^{3-5}$ in P_1. This can lead to a time gap (see D) between the WLs for different partitions. Additionally, the AERO framework stores the on-chip intermediate data between partitions into L2 buffer. The intermediate data (from $node_v^1$, see F) is then used for the computation (of $node_a^3$ and $node_v^2$) in P_1. Similar off-chip WL and on-chip intermediate data (via L2 buffer) should be performed for different partitions until the end of the full network. As a result, the inference latency can be increased to get area benefits. The L2 intermediate data transfer depends on the applied accelerator-level physical mapping strategy.

The AERO framework considers both WL-mode and WS-mode for SR constraints. However, for LR constraints, the framework focuses on WL-mode (as previously discussed) and currently does not support WS-mode. WS-mode is possible for a set of partitioned nodes in multiple image frames, but a large L2 buffer space is required to store the intermediate data of the multiple frames. This is out of the context of this paper.

D. Energy and Area estimation approach



SRAM (analytical)
CACTI
AIMC array
Network-on-Chip (DSENT)
Table-based model (post-synthesis data)
<Your Estimator>
Component Estimator Plugins
System TANIA (.YAML)
Compound Components (.YAML)
Primitive Component Library
Energy / Area Reference Table Generator
ART.yaml
Area Estimate
ERT.yaml
Energy Calculator
Workload Characterization (Mapped to Architecture)
ActionCounts (YAML)
Energy Estimate

Fig. 8. Area/Energy estimation in AERO through Accelergy [26] methodology

For area estimation of a given architectural instance and energy estimation corresponding to a specified workload, AERO integrates Accelergy [26], a component-level area and energy estimation methodology (❺ in Fig. 2). In this approach, a system architecture is hierarchically described in terms of components of various classes. For each component class, there needs to be a component-level estimator plugin (CEP) that can be queried—given the specified attributes on a component instance—for area, and energy corresponding to all actions defined on that component class. A CEP may support as fine a level of granularity for the feasible actions on a component (e.g,. read action on SRAM with the same address, or compute action on the AIMC array with specified array utilization). Accelergy also supports constructing 'compound components' from the primitive ones, the actions on which are defined in terms of actions on the constituent primitive components.

Accordingly, the TANIA architecture template (Fig. 1) is described as a set of yaml files, in terms of various components such as: the external storage for weights-banks and host-side/input-activations, inter-tile NoCs (mesh), the analog AIMC arrays of various compute-cell types, a compound component containing the SIMD lanes in a VFU (composed in turn from primitive components like integer adders/multiplers/dividers, comparators, barrel-shifter, etc.,), the intra-tile fan-out network, and the on-chip activation (L1) and L2 level SRAM buffers. While Accelergy does include a few CEPs covering primitive components (e.g., arithmetic/combinational operations, registers, etc.,) with analytical models, SRAMs modeled with CACTI, etc., considering the target process for this work (22 nm GF22FDX), and favoring more specificity, the estimation methods of some components where updated as necessary and new estimation plugins (CEPs) were added, for the components specific to Fig. 1. In particular, (highlighted in blue text in Fig. 8): an in-house model for SRAM buffers (corresponding to GF22FDX) is used in place of CACTI; a model for AIMC array (modeling this as a 'compound component' as in [27] was an option, but modeling it as a primitive with a dedicated CEP is a better choice in this case); the NoC is also modeled as a primitive component, and DSENT [28] was introduced as a new plugin; for the of arithmetic and other digital components, the existing table-based plugin is leveraged together with post-synthesis data (targeting the 22nm process and with Cadence Genus) based on RTL descriptions of respective components.

V. Case-studies

In this section, the AERO framework uses TANIA architecture template to study the impacts of: (1) WL and WS, (2) external memory access bandwidth of weights, (3) SR and LR constraints, (4) AIMC dimensions and (5) AIMC memory cell technology choices. Notice that the experimental section does not include estimating the impact of host DRAM bandwidth and NoC bandwidth[4], as they have less impact on inference latency than weight DRAM bandwidth.

A. Experimental setup

The AERO framework was developed in Python and Microsoft NNI [19] was used for exploration of the design space. The experiments were executed on a machine running *Red Hat Enterprise Linux* with Intel CPU E5-2690 at 2.60GHz configuration. ONNX format is used [18] for input workloads. Table IV lists the workloads that are considered in experiments, including a MLP (denoted by MLP-3 with 3 Fully-connected layers), Lenet-5 (a small CNN) and multiple deep Resnets (-18,-34,-50,-101). The MLP-3 [29] and Lenet-5 [30] workloads are exported to ONNX format based on their open-source code, while the Resnets are obtained from ONNX Zoo [31]. All the operations including digital operations, skip connection are modelled in the current framework. The table summarizes the number of different layers and the total weight size of each workload, considering ternary weights. In the experimental case-studies, 2 PEs are allocated to each of the first two layers (Split_2_2, recall Fig. 3 (d)) of the neural networks by default to optimize the inference latency. Split_2_2 can reduce inference latency by up to 70% with less than 13.5% increase in energy consumption (for Lenet-5 and Resnets). The relevant experimental details are not shown due to the limit space of the paper. Splitting is not performed for MLP-3, since the output dimension (*ox, oy*) of FC layer is 1. Note that the hardware technology nodes used for energy and area estimation haven been described in Section IV-D.

B. Baseline Architecture and Area Analysis

Table V presents the baseline architecture used for the experiments. Given an area budget of 200 mm^2, 4 clusters can fit when using different dimensions (Row $\in \{512, 576, 1024, 1152\}$ and Col $\in \{256, 512, 1024\}$) in each AIMC array. 1152 rows × 512 columns dimension is selected for the baseline architecture (as explained in Section V-F).

The area breakdown of the baseline architecture is presented in Fig. 9. The AIMC arrays contribute significantly (46.6%).

[4]The multi-layer parallel architecture offers the possibility to hide the latency of loading input images from host DRAM and intermediate data (*e.g.*, input, output and accumulation) via NoC.

TABLE IV
Considered Workloads

Networks	#FC	#Conv	#Pooling	NL types	Weights (MB)
MLP-3	3	0	0	Relu	0.159
Lenet-5	2	3	2	Tahn	0.015
Resnet-18	1	20	2	Relu	2.784
Resnet-34	1	36	2	Relu	5.193
Resnet-50	1	53	2	Relu	6.080
Resnet-101	1	104	2	Relu	10.596

* Ternary weights are considered for all experiments.

TABLE V
Baseline architecture parameters.

Abstraction	Hardware Resources				
Accelerator-level	4 Clusters	Host DRAM (LPDDR4, 10 Bytes/cycle)	★Weight DRAM (HBM2, 40 Bytes/cycle)		Inter-cluster & Inter-tile NoC (60 Bytes/cycle)
Cluster-level	4 Tiles	L2 buffer (512KB)			
Tile-level	4 Aicores	2 VFUs	Tile Shuffler		
PE-level	Activation Buffer	Instruction Memory	AIMC array for Aicore (1152 rows × 512 cols)		

*Digital frequency=1GHz, Analog AIMC frequency=0.1GHz

*★ is set for demonstration purpose (see Section V-D). This bandwidth is sufficient for CNNs (Lenet-5, Resnet-18), but leads to weight loading stalls for deep CNNs (Resnet-34,-50,-101).

to the overall area. The digital blocks inside the Aicores (pie in red) account for 20.9%. Activation buffers are another major contributor to area (17.5%). In modern day CNNs, the number of MAC operations dominate the total percentage of computations. For example, Resnet-18 has 3628.15 *millions* analog MAC operations and 9.65 *millions* digital operations. The area is dominated by AIMC arrays, which are responsible for performing the MAC operations that constitute the bulk of operations in the CNNs.

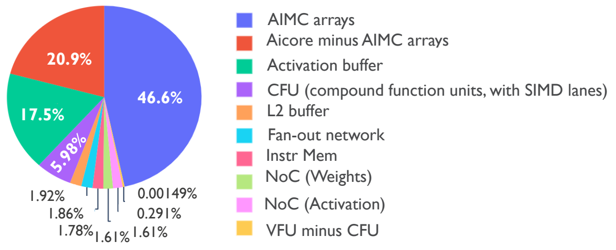


Fig. 9. Area breakdown of TANIA baseline architecture (4-cluster, 103.60 mm^2). Note: **CFU**: Compound function units (SIMD lanes).

C. Impact of weight loading and weight stationary

This case-study assesses the impact of SW and WS under SR constraints. As previously discussed in Section IV-C1, sufficient clusters allow loading weights of all layers at once. Table VI shows the resource requirements, indicating that as the application layer gets deeper (*e.g.*, from Resnet-18 to Resnet-101), the number of required clusters and corresponding area increase.

To reduce energy/latency overheads due to WL, weights can be loaded once for the first input (*e.g.*, image) and reused for the next multiple inputs (*i.e.*, denoted by *weight reuse*). When *#reuse>1*, multiple input images can be processed in parallel and the pipeline throughput is dependent on the critical path (*i.e.*, layer costing the longest computation time). When *#reuse=infinite*, the mapping can be approximated WS.

Fig. 10 compares the power efficiency (TOPs/W) and pipeline throughput (#inference/s) with increasing number of

TABLE VI
COMPARISON OF WL AND WS WITH SUFFICIENT CLUSTERS

Networks	#Clusters (SR)	Area (mm^2)	Power efficiency ($TOPs/W$)			Pipeline throughput[5] (#inference / s)		
			WL[6]	WS[7]	Ratio=WS/WL	WL	WS	Ratio=WS/WL
MLP-3	1	25.90	0.442	262.267	593.364	23618	2×10^7	846.811
Lenet-5	1	25.90	1.087	1.707	1.570	239291	283849	1.186
Resnet-18	4	103.60	29.723	80.313	2.702	10836	17173	1.585
Resnet-34	7	181.31	28.739	92.165	3.207	6415	17174	2.677
Resnet-50	8	207.21	21.077	50.562	2.399	5426	11104	2.046
Resnet-101	15	388.52	17.048	48.789	2.745	3303	11105	3.362

* Pipeline throughput is calculated at $1GHz$ based on latency cycles

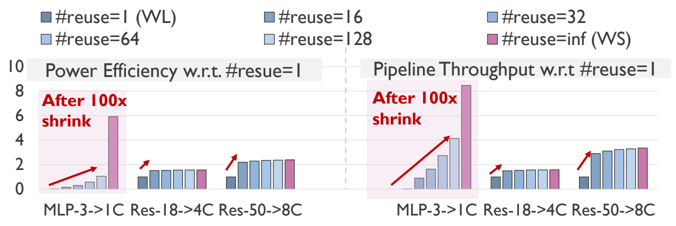


Fig. 10. (a) Normalized power efficiency (TOPs/W) and (b) Normalized pipeline throughput (#inference/s) at different #reuse, under SR constraints. Note Res-18→4C (Resnet-18 mapped to 4 cluster).

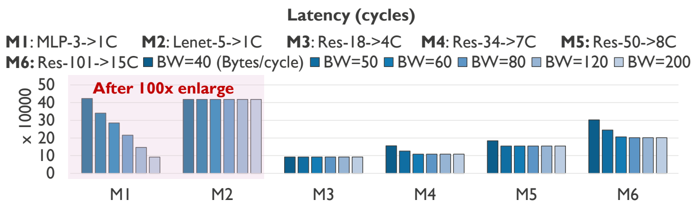


Fig. 11. Impact of external memory access of weights at increasing bandwidth (BW, Bytes/cycle) in WL mode, under SR constraints.

weight reuses. It can be observed that WL mode (*#reuse=1*) has the lowest power efficiency. MLP-3 workload (with 3 FC layers) is sensitive to *#reuse* in terms of power efficiency and performance, as FC layers are typically dominated by WL. Compared to WL mode (*#reuse=1*), WS mode (*#reuse=inf*) is $\sim$100$\times$ better in power efficiency and performance.

On the other hand, for Resnet-18,-50 (in Fig. 10), their power efficiency and pipeline throughput approaches up to 90% of WS mode (*#reuse=inf*) when #reuse=16. This indicates that *weight reuse can enable power efficiency and performance for CNNs close to WS mapping, even for low values.* Hence, the theoretical optimum can be approached with our architecture template and our mapping strategy. A broader comparison for all the considered workloads can be seen in Table VI. The table shows that WS mode has $1.570\times \sim 3.207\times$ better power efficiency and $1.186\times \sim 3.362\times$ better performance than WL mode (for CNNs, excluding MLP-3).

D. Impact of external memory access of weights

This case-study assesses the impact of external memory access bandwidth (BW) on latency in WL mode. An intuition overview can be seen Fig. 13 (a), which describes the latency behavior of Res-34→7C when BW=40 Bytes/cycle. Indexes A and B refer to the behaviors of initial layers, where WL latency is significantly smaller than computation latency. Nevertheless, possible WL stalls (index C) can defer the execution of later layers. For better performance, multiple HBMs could be interfaced to achieve higher bandwidth, similar to TPUs [32]. Fig. 11 shows the evolution of latency due to increasing weight DRAM bandwidth. It can be observed that the latency of MLP-3 is sensitive to the increase in bandwidth due to its weight-rich FC layers. Besides, the latency of Lenet-5→1C and Res-18→4C do not change with increasing bandwidth, which implies that small CNNs are dominated by computation activities (w.r.t WL). Lastly, for deep neural networks with a huge amount of weights (*e.g.*, Resnet-34,-50,-101), an increase in weight DRAM bandwidth can reduce latency. Thus, the tiled architecture can take benefits of higher available weight bandwidth to reduce the inference latency of deep CNNs. But in the end, latency saturates to a critical path, which happens around bandwidth of 60 Bytes/cycle for Resnets.

E. Impact of resource constraint

This case-study assesses the impact of SR and LR constraints. In the baseline architecture with 4 clusters, the deep Resnets (Resnet-34,-50,-101) are resource constrained. For an intuitive overview, Fig. 13 compares the latency behavior of Res-34→7C and Res-34→4C (*i.e.*, SR and LR). Under SR constraints, WL is performed for all Aicores (mapped MVM layers) in layer depth order (see index A). Under LR constraints, the virtual nodes of Resnet-34 are divided into 2 partitions (see P_0 and P_1 in part (b-c)). WL is first performed for nodes in P_0 (see index D). Once Aicore resources are released by one/some node(s) in P_0, WL can be started for P_1 (see index E, with a gap between D and E). As previously illustrated in Section IV-B, the intermediate data between two partitions are transferred via the shared L2 buffer (see F and G, corresponding to *Add-Relu_3*, *Conv_35_0* and *Conv_35_1* in part (c)). Nodes in different partitions execute sequentially until the end of the last layer.

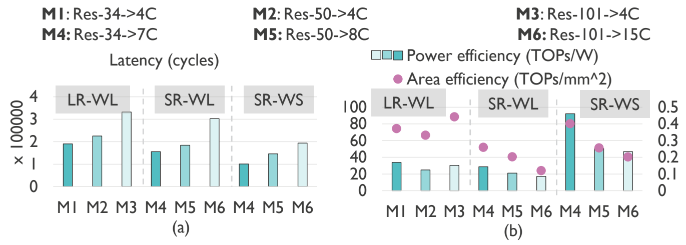


Fig. 12. (a) Latency (cycles) and (b) power efficiency (TOPs/W) and area efficiency (TOPs/mm^2) in three cases: LR-WL (*i.e.*, limited resources in weight loading mode), SR-WL and SR-WS.

Fig. 12 shows the comparison of three cases: *LR-WL* (*i.e.*, limited resources in weight loading mode), *SR-WL* and *SR-WS* for the deep Resnets (-34,-50,-101). Part (a) indicates that *LR-WL* has the highest latency (about 18% higher than SR-WL, 35% $\sim$ 47% higher than SR-WS), with area saving 43% $\sim$

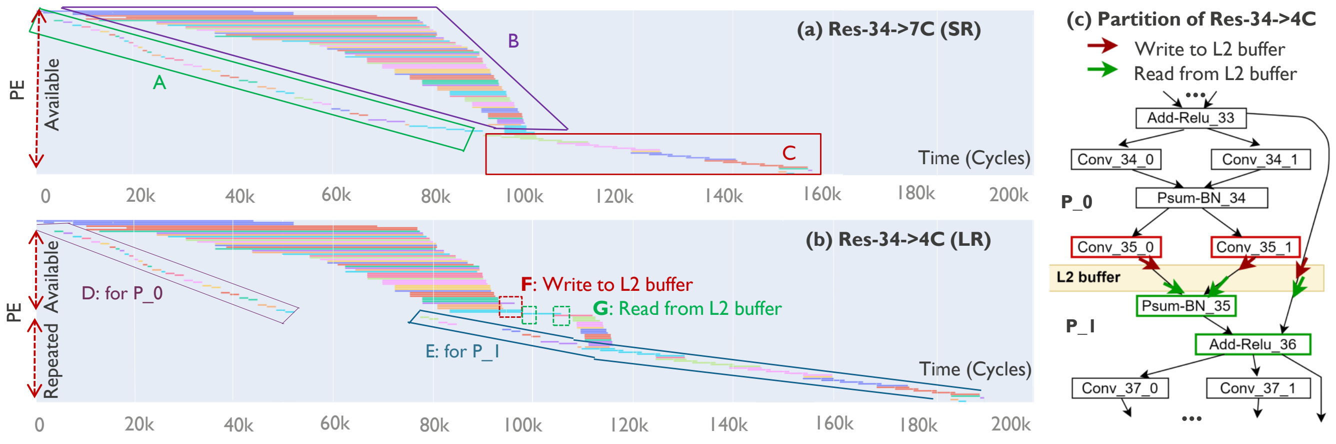


Fig. 13. Latency behavior Resnet-34 with of weight access bandwidth of 40 Bytes/cycle: (a) Res-34→7C (SR) and (b) Res-34→4C (LR). (c) Partition of Res-34→4C (*e.g.*, for *Conv_x_y*, *x* is the layer index and *y* is the mapping index on multiple Aicores (recall Section IV-A)). **Note**: **A**: weight loading (WL). **B**: computation. **C**: WL stalls. **E**: write to L2 buffer. **F**: read from L2 buffer. **G**: WL for P_0 (*e.g.*, partition index). **H**: WL and WL stalls for P_1. *Particularly for part (b), different nodes mapped to the same Aicore/VFU are displayed in different rows.

73% (4 clusters w.r.t. 7, 8 and 15 clusters). This is because in *LR* case, a full network is divided into multiple partitions. WL and intermediate data transfer (via the shared L2) are performed for each set of partitioned nodes, which leads to extract overheads (*e.g.*, latency, energy). Fig. 12 (b) shows that *SR-WS* has the highest power efficiency, followed by *LR-WL* (15% higher than *SR-WL*). Additionally, *LR-WL* has the highest area efficiency for Resnet-50,-101 (up to 54% higher than *SR-WS*). Since Resnet-34 is less resource constrained in 4 clusters than Resnet-50,-101, the area efficiency of Resnet-34 in *LR-WL* is close to (*i.e.*, 7% lower than) *SR-WS*. This proves that the AERO framework unlocks different power-area-performance exploration points under LR constraints, while LR constraints (for hybrid tile-based accelerators) are not considered in existing works.

F. Impact of AIMC dimensions

This case-study assesses the impact of AIMC dimensions. Fig.14 (a-b) shows the latency evolution of Res-18→4C (SR) and Res-50→4C (LR) across different AIMC dimensions (represented by Columns×Rows), considering WL overheads. Larger AIMC dimensions typically have lower latency, since higher output-unrolling factors (*i.e.*, more copies of weights in AIMC, recall Fig. 3 (b.3)) allow computing multiple output activations simultaneously. However, higher output-unrolling factors in a AIMC can take more time for WL. This explains why the latency of 1152×1024 is higher than 1152×512.

Fig. 14 (c-d) shows power efficiency ($TOPs/W$) and area efficiency ($TOPs/mm^2$) for the two considered networks. Good power efficiency can be achieved in large array dimension (*i.e.*, 1152 × 256, 1152 × 512, 1152 × 1024), out of which, 1152 × 512 AIMC dimension has the highest power efficiency. However, the AIMC array dimensions with largest number of rows (1024) have low area efficiency, due to severe under-utilization of array. High area efficiency is achieved in the AIMC arrays with small number of rows (*e.g.*, with 256 rows). Similar trade-offs between latency, power efficiency and area efficiency can be seen in other Resnets (*i.e.*, kernel dimension,

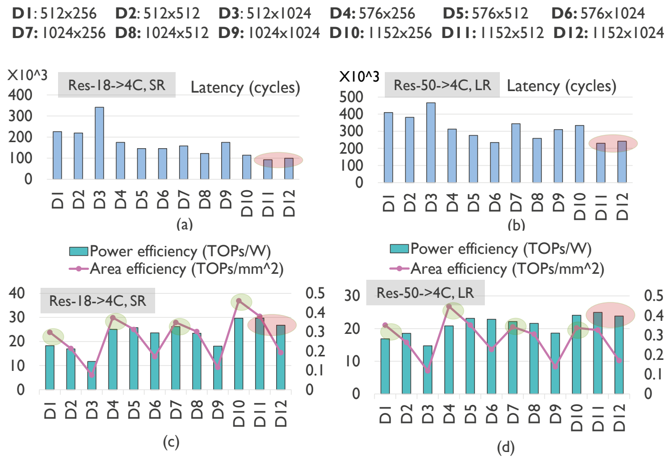


Fig. 14. (a-b) Latency evolution, (c-d) power efficiency and area efficiency of Resnet-18 and Resnet-50 mapping in 4 clusters, considering different AIMC dimensions (Columns × Rows) and WL overheads.

kx, ky = 3). On the other hand, the small networks, MLP-3 and Lenet-5, have small fluctuations in latency, power efficiency, and area efficiency in different AIMC dimensions. For MLP-3→1C, 1024 × 512 has the best trade-offs, while 1152 × 512 is the second best[5]. For Lenet-5→1C (kernel dimension, kx, ky = 5), 1024 × 1024 has the lowest latency, while 1152 × 512 has higher energy and area efficiency[6].

Overall, 1152 × 512 is selected in the baseline architecture, due to its good trade-off in latency, power efficiency and area efficiency for the considered networks in this paper. To the best of our knowledge, AERO is the only framework that explicitly reports such trade-offs, under both SR and LR constraints, on hybrid tile-based architectures.

G. Impact of AIMC memory cell technology choices

This case-study evaluates the impact of AIMC compute cell technology choices, *i.e.*, SRAM, IGZO, SOT-MRAM [17]. For

[5]**1152×512**: 4234 cycles, 0.443 $TOPs/W$ and 0.013 $TOPs/mm^2$; **1152×512**: 4234 cycles, 0.442 $TOPs/W$ and 0.012 $TOPs/mm^2$

[6]**1024×1024**: 3451 cycles, 0.878 $TOPs/W$ and 0.002 $TOPs/mm^2$; **1152×512**: 4179 cycles, 1.004 $TOPs/W$ and 0.005 $TOPs/mm^2$

TABLE VII
AIMC TECHNOLOGY CHOICES

Parameter	SRAM	IGZO	SOT-MRAM
Area	0.785 um^2	0.26 um^2	10.368 um^2
Write Energy	71118.62 pJ	686909.03 pJ	1191501.10 pJ
Compute Energy	1064.98 pJ	693.16 pJ	1655.76 pJ
Write Latency	1 ns	10 ns	3 ns [33]
Compute Latency	2.3 ns	0.5 ns	2 ns [34]

*Write energy is for writing a row of cells. Compute energy is for MVM operations in one AIMC cycle (10 ns, 0.1GHz in the baseline architecture).
*Compute latency of SOT-MRAM is in 45nm node [34], while other parameters are in 22nm node (in-house data). The largest parameter in each row is highlighted.

the three technology choices, Table VII compares their area, energy and latency parameters, showing clear Pareto trade-offs in the 3 dimensional space.

In the baseline architecture, the digital frequency (f_D) and analog AIMC frequency (f_A) are set to 1 GHz and 0.1 GHz, respectively (see Table V). In our assumptions, when the AIMC write latency (L_W, affecting the speed of WL) is the same as the digital cycle (1 ns), there is no WL stall due to AIMC write (*e.g.*, for SRAM). The higher L_W of IGZO and SOT-MRAM results in additional WL stalls at f_D, and their WL speeds are 10× and 3× slower than SRAM, respectively. On the other hand, AIMC compute latency (L_C) limits the maximum analog frequency ($max(f_A) = \frac{f_D}{L_C}$) and has a large impact on the overall latency for a given inference workload. In WS mode, ideally, the inference latency of the three technologies is approximately proportional to their L_C. Considering DAC/ADC requirement between analog and digital components, this case-study fixes $f_A = 0.1GHz$ (with 10 ns analog cycle). Since the L_C of the three technologies are within the analog cycles, the compute latency of SOT-MRAM (45nm node [34]) caused by different technologies (w.r.t. 22nm node, see Table VII) does not affect the AIMC computation nor the experimental results.

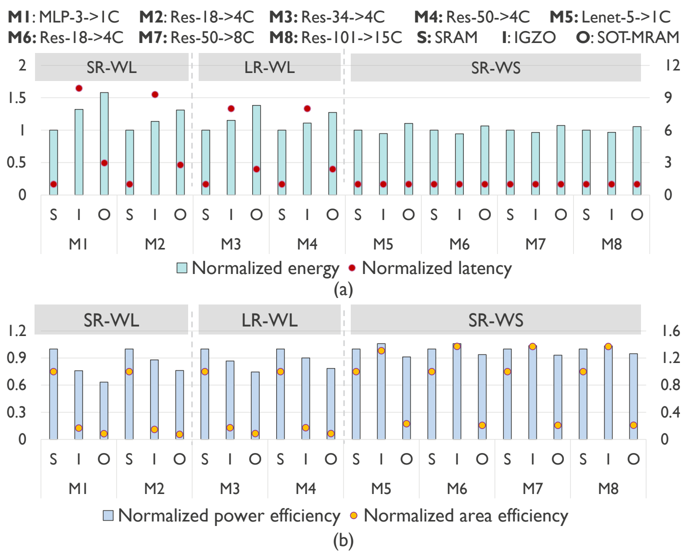


Fig. 15. (a) Normalized energy and latency, along with (b) normalized power efficiency (TOPs/W) and area efficiency (TOPs/mm^2) for SRAM (S), IGZO (I) and SOT-MRAM (O) AIMC technology choices.

Fig. 15 compares normalized energy and latency, power efficiency and area efficiency (w.r.t. SRAM) for different AIMC technologies. Two observations can be made.

- In WL mode (SR-WL and LR-WL, under SR and LR constraints), SRAM leads to the lowest energy and lowest latency. This is because SRAM has the minimum AIMC write energy and write latency. Consequentially, SRAM has the highest power efficiency and area efficiency.
- In WS mode, IGZO results in the lowest energy, due to its lowest compute energy. The latency is comparable across the three AIMC technologies. Besides, IGZO has the highest power efficiency and area efficiency.

VI. Discussion

This section discusses the advantages and limitations of the proposed AERO DSE framework. As presented in the previous sections, AERO is able to assess the impacts of several design parameters and derive insights into tile-based hybrid accelerators, such as:

- Low values of weight reuse (*e.g.*, 16) can enable energy efficiency and performance for CNNs close to weight stationary mapping.
- Higher weight access bandwidth can reduce the inference latency of deep CNNs. But in the end, latency saturates to a critical path, which happens around 60 Bytes/cycle.
- *LR-WL* is a power-area-performance trade-off solution. It has 18% higher latency than *SR-WL*, with 15% higher power efficiency and 73% lower area. Compared to *SR-WS*, *LR-WL* has close or even 54% higher area efficiency.
- The 1152 col×512 row AIMC has a good trade-off in inference latency, energy efficiency and area efficiency for Resnets as well as Lenet-5 and MLP-3.
- For AIMC cell technologies (SRAM, IGZO and SRAM), SRAM has lowest energy/latency in WL mode, while IGZO has the best power/area efficiency in WS mode.

TABLE VIII
SUPPORTED DSE OPTIONS IN AERO

Abstraction	DSE options
Accelerator-level	#Clusters; NoC design / technology parameters; Instr Mem Dim (**E, A**); Bandwidths of host & weight bank & NoC (**L**)
Cluster-level	#Tiles in a cluster; L2 buffer dimensions
Tile-level	Aicores-VFU organization in a tile; Actbuf Dim
PE-level	#Rows & #Colums of AIMC; AIMC cell technology; ADC & DAC precision; AIMC & VFU compute speed
Mapping-level	Fusion operation order; Splitting factor configuration

* **Instr Mem**: Instruction memory, **Actbuf**: Activation buffer, **Dim**: dimension, **L**: Latency-only, **E**: Energy-only, **A**: Area-only

Further, AERO supports a broad range of DSE options to facilitate co-exploration of mapping, architecture and technology decisions. Table VIII summarizes the currently supported DSE options in AERO.

The current version of AERO has two main limitations. First, latency estimation at the NoC level does not model communication contention. Second, loop transformation exploration over the memory hierarchy is not supported. It must be noted, however, that existing loop transformation DSE tools (*e.g.*, Timeloop [8], Interstellar [9], Zigzag [10] for single-layer parallel architecture) are too pessimistic for multi-layer parallel architectures as they do not take into account the dynamic data-flow between two neighboring layers.

VII. Conclusion

This paper presents a DSE framework, AERO, for a hybrid digital-analog CNN accelerator architecture that supports multi-layer parallel execution. The AERO framework presents a general mapping flow, i ncluding v irtual m apping (allowing node fusion, node splitting configurations) a nd physical mapping (a hierarchical strategy) to fulfil t he m apping at accelerator-level, cluster-level, tile/PE-level and AIMC-level. Besides, AERO incorporates detailed PPA models to characterize the latency, energy and area of a full neural network executing under sufficient a nd l imited r esource constraints. The experimental case-studies on MLP, Lenet-5 and Resnets (-18,34,50,101) derive insights of design parameters, such as weight loading / weight stationary, and sufficient / limited resource constraints, on the tile-based hybrid architectures.

References

[1] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 609–622, 2014.

[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] Shimeng Yu, Hongwu Jiang, Shanshi Huang, Xiaochen Peng, and Anni Lu. Compute-in-memory chips for deep learning: Recent trends and prospects. *IEEE Circuits and Systems Magazine*, 21(3):31–56, 2021.

[4] Pouya Houshmand, Stefan Cosemans, Linyan Mei, Ioannis Papistas, Debjyoti Bhattacharjee, Peter Debacker, Arindam Mallik, Diederik Verkest, and Marian Verhelst. Opportunities and limitations of emerging analog in-memory compute dnn architectures. In *2020 IEEE International Electron Devices Meeting (IEDM)*, pages 29–1. IEEE, 2020.

[5] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. *ACM SIGARCH Computer Architecture News*, 44(3):14–26, 2016.

[6] Xiaochen Peng, Rui Liu, and Shimeng Yu. Optimizing weight mapping and data flow for convolutional neural networks on rram based processing-in-memory architecture. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2019.

[7] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 715–731, 2019.

[8] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek 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.

[9] 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.

[10] Linyan Mei, Pouya Houshmand, Vikram Jain, Sebastian Giraldo, and Marian Verhelst. Zigzag: Enlarging joint architecture-mapping design space exploration for dnn accelerators. *IEEE Transactions on Computers*, 2021.

[11] Martino Dazzi, Abu Sebastian, Luca Benini, and Evangelos Eleftheriou. Accelerating inference of convolutional neural networks using in-memory computing. *Frontiers in Computational Neuroscience*, page 63, 2021.

[12] 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.

[13] Joao Ambrosi, Aayush Ankit, Rodrigo Antunes, Sai Rahul Chalamalasetti, Soumitra Chatterjee, Izzat El Hajj, Guilherme Fachini, Paolo Faraboschi, Martin Foltin, Sitao Huang, et al. Hardware-software co-design for an analog-digital accelerator for machine learning. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–13. IEEE, 2018.

[14] Tvm: Open deep learning compiler stack. https://github.com/apache/tvm. Accessed: 2021-12-01.

[15] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. {TVM}: An automated end-to-end optimizing compiler for deep learning. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 578–594, 2018.

[16] Debjyoti Bhattacharjee, Nathan Laubeuf, Stefan Cosemans, Ioannis Papistas, Arindam Mallik, Peter Debacker, Myung Hee Na, and Diederik Verkest. Design-technology space exploration for energy efficient aimc-based inference acceleration. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2021.

[17] S Cosemans, B Verhoef, J Doevenspeck, IA Papistas, F Catthoor, P Debacker, A Mallik, and D Verkest. Towards 10000tops/w dnn inference with analog in-memory computing–a circuit blueprint, device options and requirements. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 22–2. IEEE, 2019.

[18] Open neural network exchange. https://onnx.ai/.

[19] Neural network intelligence. https://www.microsoft.com/en-us/research/project/neural-network-intelligence/. Accessed: 2021-11-09.

[20] Manoj Alwani, Han Chen, Michael Ferdman, and Peter Milder. Fused-layer cnn accelerators. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–12. IEEE, 2016.

[21] Hakan Aydin and Qi Yang. Energy-aware partitioning for multiprocessor real-time systems. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 9–pp. IEEE, 2003.

[22] Wei Quan and Andy D Pimentel. A hierarchical run-time adaptive resource allocation framework for large-scale mpsoc systems. *Design Automation for Embedded Systems*, 20(4):311–339, 2016.

[23] Maximilian Götzinger, Amir M Rahmani, Martin Pongratz, Pasi Liljeberg, Axel Jantsch, and Hannu Tenhunen. The role of self-awareness and hierarchical agents in resource management for many-core systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 53–60. IEEE, 2016.

[24] Metis - serial graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.

[25] Inman Harvey. The microbial genetic algorithm. In *European conference on artificial life*, pages 126–133. Springer, 2009.

[26] Yannan N. Wu, Joel S. Emer, and Vivienne Sze. Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs. In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2019.

[27] Yannan Nellie Wu, Vivienne Sze, and Joel S Emer. An architecture-level energy and area estimator for processing-in-memory accelerator designs. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 116–118. IEEE, 2020.

[28] Chen Sun, Chia-Hsin Owen Chen, George Kurian, Lan Wei, Jason Miller, Anant Agarwal, Li-Shiuan Peh, and Vladimir Stojanovic. Dsent-a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, pages 201–210. IEEE, 2012.

[29] Multi-layer perceptron, MNIST. https://github.com/iam-mhaseeb/Multi-Layer-Perceptron-MNIST-with-PyTorch/blob/master/mnist_mlp_exercise.ipynb.

[30] Lenet-5. https://github.com/activatedgeek/LeNet-5/blob/master/lenet.py.

[31] Onnx model zoo. https://github.com/onnx/models.

[32] High-bandwidth memory (hmb). https://www.amd.com/system/files/documents/high-bandwidth-memory-hbm.pdf. Accessed: 2021-12-01.

[33] Md Rubel Sarkar, Md Maruf Abir Bappy, Md Minhajul Azmir, Dewan Mohammed Rashid, and Saad Ibn Hasan. Vg-sot mram design and performance analysis. In *2021 IEEE 12th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 0715–0719. IEEE, 2021.

[34] Zhezhi He, Yang Zhang, Shaahin Angizi, Boqing Gong, and Deliang Fan. Exploring a sot-mram based in-memory computing for data

processing. *IEEE Transactions on Multi-Scale Computing Systems*, 4(4):676–685, 2018.

Simei Yang is currently a postdoc at IMEC and KU Leuven. She received her Ph.D. degree in the embedded system in IETR UMRCNRS 6164- Polytech Nantes - Université de Nantes, France, in 2020. Prior to this, she received the B.Sc. and M.Sc. degrees in microelectronics (Integrated Circuit Engineering) from Sun Yat-sen University, Guangzhou, China, in 2014 and 2016, respectively. Her research interests include DNN evaluation framework on compute-in-memory based accelerators, run-time management of energy efficiency on multi/many-core systems, and system-level modelling and simulation.

Debjyoti Bhattacharjee is a Research and Development engineer at Compute System Architecture unit at imec, Leuven. He received the BTech degree in computer science and engineering from the West Bengal University of Technology (WBUT), Kolkata, West Bengal, India, in 2013, the MTech degree in computer science from the Indian Statistical Institute, Kolkata, India, in 2015, and the PhD degree in computer science and engineering from the Nanyang Technological University, Singapore, in 2019. He worked as a research fellow with Nanyang Technological University, Singapore for a year. During his doctoral studies, he worked on design of architectures using emerging technologies for in-memory computing. He developed novel technology mapping algorithms, technology-aware synthesis techniques, and proposed novel methods for multi-valued logic realization.

His current research interests include machine learning accelerator using analog hardware, hardware design automation tools and application-specific accelerator design, with emphasis on emerging technologies.

Vinay B. Y. Kumar received a Ph.D. in EE from the Indian Institute of Technology, Bombay, in 2019, and also the B.Tech + M.Tech (Microelectronics) degrees from the same institute in 2008. He is presently a researcher and a project technical lead at IMEC, Leuven, with a focus on future compute systems modeling. Prior to this, he was a post-doctoral research fellow, for 2 years, at the School of Computer Science and Engineering in Nanyang Technological University (Singapore) where he contributed to the SOCure project, a programme towards secure SoC design. Prior to Ph.D., he was at the Davinci Innovation Center at ASUS, in Taipei, where he worked on applied speech processing.

Saikat Chatterjee is a research engineer at IMEC Leuven, Belgium. His research focuses on computer architecture, asic design, block and chip level verification. Saikat received his PhD in Intelligent Systems from Bielefeld University, Germany, MS in Instrumentation from Indian Institute of Science (IISc) Bangalore, India and BE in Instrumentation and Electronics Engineering from Jadavpur University (JU), India.

Sayandip De is a researcher at IMEC Leuven, Belgium and guest research candidate in the Electronic Systems group at Eindhoven University of Technology (TU/e). His research focuses on computer architecture, approximate computing, design automation for low power circuits and systems, optimizing machine learning inference. Sayandip received his M.Tech in VLSI Design from Indian Institute of Engineering Science and Technology (IIEST) Shibpur, India. He received his B.Tech in Electronics and Communication Engineering from Kalyani Government Engineering College (KGEC), India.

Peter Debacker received the M.Sc. (Hons.) degree in electrical engineering from the Katholieke Universiteit Leuven, Leuven, Belgium, in 2004. He worked with Philips as a system engineer and at Essensium as a System Architect before joining IMEC, Leuven, in 2011.

At imec he is currently Program and leads a team that researches architecture, algorithms and circuits and devices to create efficient AI hardware ranging from large and tiny DNN accelerators to in-memory compute and neuromorphic hardware.

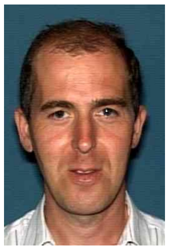
Before that, he has worked on IMEC's low-power digital chip and processor architectures and implementation in advanced technology nodes to optimize power-performance-area (PPA) optimization of scaled CMOS technologies (for 3nm and beyond).

His current research interests include AI, machine learning and neuromorphic computing, processor and computer architectures, design methodologies, design-technology co-optimization, digital chip design. He was co-chair of the tinyML EMEA Technical forum 2021 and will chair the 2022 tinyML EMEA forum.

Diederik Verkest holds a Ph.D. in Applied Sciences from the University of Leuven, Belgium. He started working in the VLSI design methodology group of imec Leuven, Belgium) on hardware/software co-design, re-configurable systems, and multi-processor system-on-chips in the domain of wireless and multimedia. From 2010 to 2020 he was responsible for imec's Logic Insite research program in which the leading design and process companies jointly work on co-optimization of CMOS design and process technology for N+2 nodes. In recent years he focused on design and process technology optimization for ML accelerators.

Diederik Verkest published and presented over 150 articles in international journals and at international conferences.

Arindam Mallik leads the Future System Exploration (FuSE) group in the Compute System Architecture (CSA) RD unit. He received M.S, and PhD degree in Electrical Engineering and Computer Science from Northwestern University, USA in 2004 and 2008, respectively. Arindam is a technologist with 20 years of experience in semiconductor research. He has authored or co-authored more than 100 papers in international journals, conference proceedings, and holds number of international patents. His research interests include novel computing system, design-technology co-optimization, economics of semiconductor scaling.



Francky Catthoor received a Ph.D. in EE from the Katholieke Univ. Leuven, Belgium in 1987. Between 1987 and 2000, he has headed several research domains in the area of synthesis techniques and architectural methodologies. Since 2000 he is strongly involved in other activities at IMEC including co-exploration of application, computer architecture and deep submicron technology aspects, biomedical systems and IoT sensor nodes, and photo-voltaic modules combined with renewable energy systems, all at IMEC Leuven, Belgium. Currently he is an IMEC senior fellow. He is also part-time full professor at the EE department of the KULeuven.

He has been associate editor for several IEEE and ACM journals, and was elected IEEE fellow in 2005.