A Uniform Latency Model for DNN Accelerators with Diverse Architectures and Dataflows

Linyan Mei*†, Huichu Liu*, Tony Wu*, H. Ekin Sumbul*, Marian Verhelst†, Edith Beigne*
*Meta Reality Labs, †MICAS-ESAT, KU Leuven
{huichu, tonyfwu, ekinsumbul, Edith.Beigne}@fb.com, {linyan.mei, marian.verhelst}@kuleuven.be

***Abstract*—In the early design phase of a Deep Neural Network (DNN) acceleration system, fast energy and latency estimation are important to evaluate the optimality of different design candidates on algorithm, hardware, and algorithm-to-hardware mapping, given the gigantic design space. This work proposes a uniform intra-layer analytical latency model for DNN accelerators that can be used to evaluate diverse architectures and dataflows. It employs a 3-step approach to systematically estimate the latency breakdown of different system components, capture the operation state of each memory component, and identify stall-induced performance bottlenecks. To achieve high accuracy, different memory attributes, operands' memory sharing scenarios, as well as dataflow implications have been taken into account. Validation against an in-house taped-out accelerator across various DNN layers has shown an average latency model accuracy of 94.3%. To showcase the capability of the proposed model, we carry out 3 case studies to assess respectively the impact of mapping, workloads, and diverse hardware architectures on latency, driving design insights for algorithm-hardware-mapping co-optimization.**

***Index Terms*—DNN accelerator, latency, cycle count, cost model, analytical model, dataflow, mapping**

I. Introduction

The success of DNNs in various machine learning tasks has given rise to wide adoption of specialized DNN accelerators in embedded systems for edge intelligence [1]. For real-time and resource-constrained applications, achieving performant and efficient DNN acceleration is challenging, as it requires a close co-optimization of hardware architectures with algorithms and algorithm-to-hardware mapping, a.k.a. dataflow [2]. Since exhaustive physical simulations/implementations lack intuition and are impractical to explore the large number of design choices in terms of algorithm, hardware and mappings (AHM), fast and accurate energy and latency estimation are required for early-phase Design Space Exploration (DSE).

Many prior arts have proposed uniform energy models for DNN accelerator design [3]–[9]. The common basis is an analytical model which counts the operations of each hardware component (e.g., memory read and write at each level, multiply-accumulate (MAC), data transfer in NoCs, etc.), and multiply these with the corresponding unit energy to obtain the total system energy. Machine-learning (ML) and regression based energy models trained with simulation or testing data have also been introduced [10]. All of the above methods have reported good energy estimation accuracy with fast evaluation speed.

Unlike the well-explored energy models, analytical latency models are, however, less systematically developed or explained for DNN accelerators. In the paper, we refer to "latency" as the clock cycle count (CC) for completing a workload. From the physical level to system abstraction, the existing SotA latency estimation methods can be categorized as: 1) measurement on physical device, 2) FPGA emulation [11], 3) RTL simulation [12], 3) cycle-accurate simulation [13], [14], 4) regression or ML-based methods [15], [16], and 5) analytical modeling [4], [6], [17]. Among these, analytical models are preferred for early-phase DSE, thanks to their fast run-time (orders of magnitude faster than others) and transparency for analyzing the operation of each hardware component (compared to black-box regression or ML based methods). Moreover, since most of the DNNs are deterministic with pre-known hardware architectures and mappings, high accuracy in latency is analytically achievable.

However, most existing analytical latency models rely on ideal assumptions, such as: 1) all memories at different levels are double-buffered, assuming that double-buffering can avoid all temporal stall; 2) memories that are shared by multiple operands always have multiple read/write ports to avoid interference among different operands' data accesses. Although these assumptions simplify the modeling, two issues are introduced: 1) for fixed architectures, modeling accuracy degrades if not meeting these assumptions; 2) for architecture search, memory system overhead is introduced by default that excludes a large part of design space, leading to sub-optimality. Some other latency models were delicately built for a specific design case or for a small group of design variants with a hardware template [16]. Although accuracy is preserved, the limitation in generality prevents its usage for novel architecture search.

This work aims to bridge this gap by a uniform analytical latency modeling approach for AHM co-optimization, tageting on dense intra-layer cases. The paper's key contributions are:

- Provide a comprehensive overview of the latency impact factors and introduce our modeling philosophy for solving the modeling challenges (Section II).
- Present the proposed novel analytical latency model in detail; Highlight the 3-step approach that can uniformly address the multi-level memory system induced different stall scenarios (Section III).
- Demonstrate high model accuracy with testchip validation (Section IV); Assess AHM-latency co-optimization through 3 case studies (Section V) to drive design insights.

II. AHM and Latency

A. Latency Impact Factors

1) Algorithm (A): This includes DNN layer parameters, such as layer type (e.g., Conv2D, Dense, Depthwise and Pointwise), layer loop dimensions, data attributes of the layer operands (e.g., total data size and data precision), etc.

This work is done during Linyan Mei's internship at Meta (formerly the Facebook Company).

2) Hardware Architecture (H): A DNN accelerator is usually equipped with a MAC array and a multi-level memory system, connected via an on-chip network. Its performance roofline is determined by hardware parameters, such as MAC array size, interconnectivity, and memory hierarchy (e.g., memory levels, capacity / bandwidth (BW) / number of read/write ports, and memory allocation for different operands).

3) Mapping (M): Dataflow (a.k.a. mapping) determines how the algorithm is spatially and temporally mapped on the hardware. Spatial mapping defines how to parallelize DNN loops across the MAC array, while temporal mapping defines in what order the MAC array processes the non-spatially-unrolled DNN loops. Ideal spatial mapping fully utilizes the MAC array, while ideal temporal mapping maximize operands' data reuse at lower memory levels. Mapping optimization can help to minimize compute cycle count and communication stalls.

These three factors are strongly interrelated and form a gigantic AHM design space, in which each point corresponds to a specific algorithm-hardware-mapping scenario with a resulting deterministic latency value.

B. Challenges in Uniform Latency Modeling

The challenges for building an analytical latency model that can be applied to the vast AHM design space are twofold:

First, the concurrency and interference of data transfer between different memory levels for different operands, i.e., weight (W) / input (I) / output (O), needs to be captured. Such interference comes from hardware constraints (e.g., insufficient memory BW / ports, lack of double buffering, system interrupts), and mapping choices (i.e., optimized mappings can alleviate the interference while non-optimized mappings may aggravate such effect). Note that this is specific for analytical latency estimation, but usually less impactful on analytical energy modeling. This is because analytical energy model (dynamic energy) only relies on total operation count of each hardware component, while latency also depends on when these operations happen and how they interfere with each other.

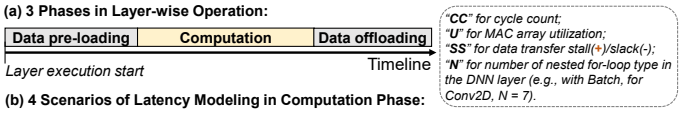
The second challenge stems from the generality requirement, since the latency model needs to be applicable for not only few pre-defined hardware architectures with a fixed dataflow, but also for every valid AHM point in the design space.

C. Proposed Modeling Philosophy

Our proposed latency modeling approach aims to solve these challenges: 1) To capture the concurrency and interdependencies of data transfers, we start from dividing this complex intertwining problem into multiple single-component data movement events, analyze each event separately, and then combine them based on actual hardware design constraints; 2) To ensure generality, we adopt a uniform AHM representation, and implement a standard 3-step memory-type / bandwidth / sharing-aware latency modeling methodology which can cover all valid design points, as detailed in Section III.

III. A Uniform Intra-Layer Latency Modeling Methodology

The processing of each DNN layer consists of 3 phases: data pre-loading, computation, and data offloading, as shown in Fig. 1(a). We define the data pre-loading as the data initialization step before computation starts, and the data offloading



(b) 4 Scenarios of Latency Modeling in Computation Phase:

Scenarios	Is MAC array **temporally** fully mapped?	Is MAC array **spatially** fully mapped?	Latency [cycle count]	MAC Array Utilization [%]
①	Yes	Yes	$CC_{ideal} = \frac{Total\ MAC\ Ops}{MAC\ array\ size}$	$U_{ideal} = 100\%$
②	Yes	No *	$CC_{spatial} = \prod_{i=1}^{N} \left\lceil \frac{loop_i\ dim.\ size}{loop_i\ unroll\ size} \right\rceil$ ***	$U_{spatial} = \frac{CC_{ideal}}{CC_{spatial}}$
③	No **	Yes	$CC_{temp.} = CC_{ideal} + SS_{overall}$ *** (> 0)	$U_{temp.} = \frac{CC_{ideal}}{CC_{temp.}}$
④	No **	No *	$CC = CC_{spatial} + SS_{overall}$ (> 0)	$U = \frac{CC_{ideal}}{CC}$

* **Reason:** MAC array / interconnects / spatial mapping and DNN layer dimension do not match, etc.
** **Reason:** Memory BW and # of port are in-sufficient; mapping is non-ideal; control overhead; etc.
*** **Define**: $\underline{Spatial\ stall} = CC_{spatial} - CC_{ideal}$; $\underline{Temporal\ stall} = CC_{temp.} - CC_{ideal} (= SS_{overall})$.

Fig. 1. (a) A timeline illustration of DNN layer operation phases. (b) Four scenarios of latency and utilization modeling in the computation phase.

as the final round of outputs writing back to memory after computation finishes. We can derive their latency based on the required data transfer amount and the related memories' BW.

The computation phase usually dominates the overall processing time, of which the latency is strongly impacted by AHM. Fig. 1(b) shows the 4 computation scenarios based on the spatial and temporal mapping rate of the MAC array. The latency modeling's challenges mainly come from the ones with temporally under-utilized MAC array (③④), due to the complexity to model the stall induced by the non-ideal data movement, i.e., temporal stall ($SS_{overall}$).

In the remaining section, we first introduce prerequisite concepts, then describe in detail the 3-step latency modeling methodology to address $SS_{overall}$ modeling challenge. The key terminologies and the model steps are illustrated in Fig. 2. Please refer to it for all the abbreviations used in this section.

A. Prerequisite Concepts and Terminology

To simplify the explanation, we adopt the data representation and loop characterization from ZigZag [8]: 1) a DNN layer is presented as a 7-dimentional nested for-loop format, namely, batch (B), output channel (K), input channel (C), output x-y dimension size (OX/OY), and filter x-y dimension size (FX/FY); 2) three major operands (W/I/O) each have their relevant (r) and irrelevant (ir) for-loops, where r / ir loops contribute to that operand's data size / data reuse respectively. For example, W's r loops are {K, C, FX, FY}, and its ir loops are {B, OY, OX}. In addition, we introduce the following terms described in the table of Fig. 2(a) and refer to them in modeling: $Unit\ Mem$, DTL, Mem_{DATA}, Mem_{CC}, and $RealBW$.

B. Step 1: Divide memory system into multiple Unit Memories by operand and compute each DTL's attributes

In the first step, we divide the problem of extracting the total temporal stall cycles $SS_{overall}$ of the entire memory system into deriving the stall/slack cycles (SS_u) induced by a single operand (W/I/O) accessing a Unit Mem (e.g., Mem1-9 in Fig. 2(b)). To analyze these Unit Mem levels, we decouple read and write operations on the interface between two Unit Mem levels, each as a DTL (e.g., from ① to ⑱ in Fig. 2(b)).

1) Compute $ReqBW_u$: For each DTL, $ReqBW_u$ is defined as the minimum memory BW to allow computation to proceed without stall. Based on the memory type (single- or double-buffered) and the top temporal loop type allocated to that

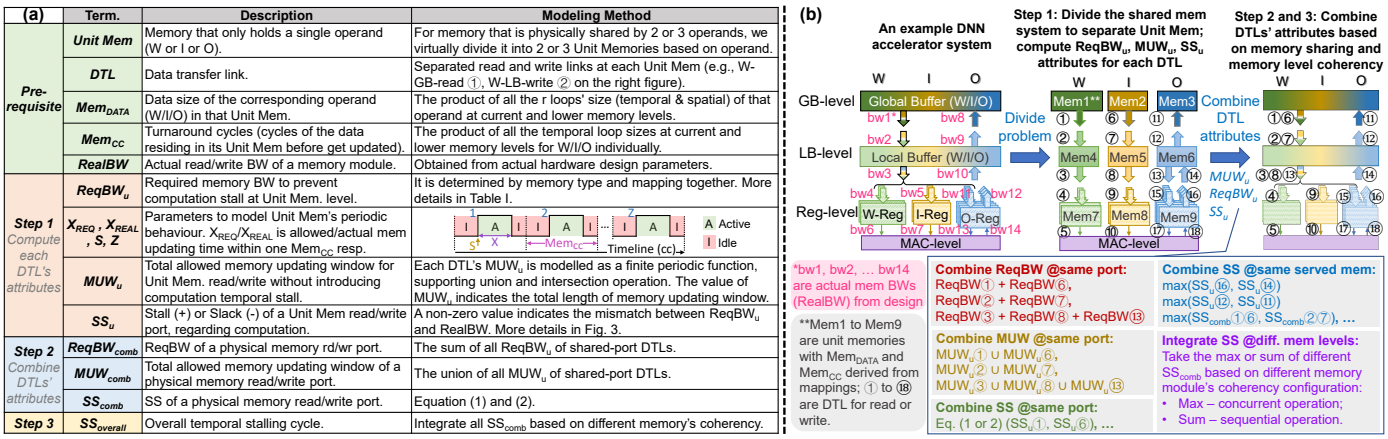


Fig. 2. (a) Descriptions of the terminologies used by each step in the proposed latency model and (b) a flow illustration of 3-step latency modeling methodology.

memory level, this parameter can be derived as shown in Table I. It is worth noting that for non-double-buffered memory levels, if an ir loop is scheduled as the upper for-loop of that level, the required data cannot be overwritten during processing that top ir loop due to data reuse (an example to visualize this is given later in Fig. 4). Thus, to avoid stall, this minimum BW requirement needs to be scaled up by all top ir loop sizes. Accordingly, $ReqBW_u$ can be obtained for each DTL.

TABLE I
$ReqBW$ DETERMINED BY BOTH MEMORY TYPE AND MAPPING.

Memory type	DB* mem.	Non-DB dual-port mem.	
Top temporal loop type	r or ir	r	ir
Physical mem capacity	A	A	A
Mapper-seen capacity	½ × A	A	A
ReqBW	BW_0**	BW_0	BW_0 × top-ir loop size

* DB = Double-Buffered ** $BW_0 = Mem_{DATA}/Mem_{CC}$

2) Derive the Unit Mem's operation pattern and memory updating window (MUW_u): When no interference occurs, the periodic pattern of memory operation can be modeled by a periodic function with 4 parameters as shown in Fig. 2(a): period (Mem_{cc}), active cycle count in one period (X), active cycle starting point in one period (S), and total number of periods (Z). Here X_{REQ} is the maximum allowed memory updating window for no-stall scenario within one Mem_{cc}, which equals to $Mem_{DATA}/ReqBW$. Accordingly, $MUW_u = X_{REQ} \times Z$. On the other hand, the actual memory BW ($RealBW$) is hardware design specific (Fig. 2(b)), hence we use X_{REAL} to represent the actual memory updating window within one Mem_{cc}, which equals to $Mem_{DATA}/RealBW$.

3) Extract SS_u: For each DTL, SS_u measures the relative cycle difference between "memory operation" and "computation", and is computed by $SS_u = (X_{REAL} - X_{REQ}) \times Z$. Fig. 3 shows SS_u visualization with six cases of computation and memory operation timelines. E.g., Fig. 3(a)(d) have $SS_u = 0$ since $X_{REAL} = X_{REQ}$, despite their different memory types;

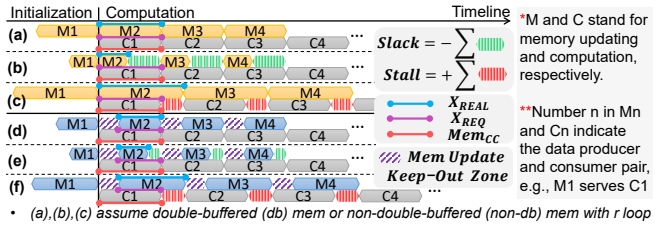


Fig. 3. Six different timeline cases of computation (C) and memory update (M) showing memory induced stall/slack for a single DTL.

same principle can be applied to obtain (b)(e)'s negative SS_u and (c)(f)'s positive SS_u.

C. Step 2: Combine the attributes on DTLs that share same physical memory port and serve same memory module

With the basic attributes $ReqBW_u$, MUW_u and SS_u obtained for each DTL in Step 1, we can derive $ReqBW_{comb}$, MUW_{comb} and SS_{comb} for DTLs that share the same physical memory port and for DTLs that serve the same memory.

1) Derive $ReqBW_{comb}$: For DTLs that share one physical memory port, the $ReqBW_{comb}$ is the sum of all the DTL's $ReqBW_u$ on that port, but with read and write distinguished.

2) Derive SS_{comb}: We first compute the union of all MUW_u for shared-port DTLs as MUW_{comb}, then calculate SS_{comb} based on the polarity (+/-) of SS_u:

- Case 1: all $SS_u \leq 0$ (assume n DTLs sharing one port):
$$SS_{comb} = \sum_{i=1}^{n}(MUW_u(i) + SS_u(i)) - MUW_{comb} \quad (1)$$
- Case 2: at least one $SS_u > 0$ (assume $SS_u(i) > 0$, $when\ 1 \leq i < m$; $SS_u(i) \leq 0$, $when\ m \leq i \leq n$):
$$SS_{comb} = \sum_{i=1}^{m-1} SS_u(i) + max(0, \sum_{i=m}^{n}(MUW_u(i) + SS_u(i)) - MUW_{comb}) \quad (2)$$

where $SS_u(i)$ is the SS_u of the i_{th} DTL; $MUW_u(i)$ is the MUW_u of the i_{th} DTL. Eq. (1) indicates that if all the shared-port DTLs do not introduce stall individually, the combined stall is the sum of each DTL's active cycles minus the maximal allowed memory updating window (MUW_{comb}). This value can be positive (generate stall) or non-positive (no stall). Eq. (2) indicates that if some of the DTLs by themselves already introduce stall (positive SS_u), we firstly combine the rest non-positive stalls using Eq. (1). If this combined result is positive, it is then added to the sum of the positive SS_u to obtain SS_{comb}; otherwise, only the sum of the positive SS_u is used as SS_{comb}. This ensures the stall(+) induced by individual DTL is not cancelled by other DTL's slack(-) during combination.

The next step is to further combine the SS of the DTLs serving the same memory. This final SS_{comb} is the maximal value either out of their SS_u (e.g., $max(SS_u$⑫, SS_u⑪) in Fig. 2(b)) or out of the already combined SS_{comb} (e.g., $max(SS_{comb}$①⑥, SS_{comb}②⑦) in Fig. 2(b)).

3) A detailed example: To visualize Step 1 and 2, an example is given in Fig. 4. It puts 5 design factors together: 1) hardware architecture, 2) mapping, 3) loop iteration, 4) data access pattern, and 5) memory-compute timeline, and illustrate how these factors interact with each other and impact latency.

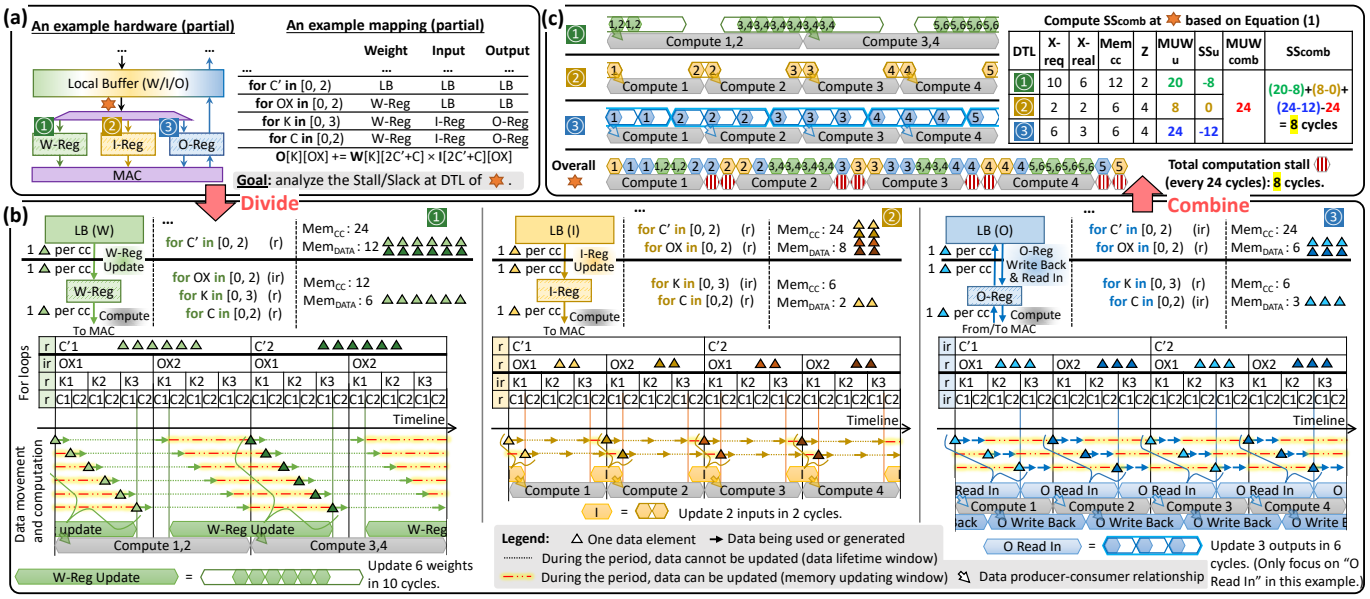


Fig. 4. An example demonstration of (a)-(b) Step 1 (Divide) and (b)-(c) Step 2 (Combine) for deriving the intermittent modeling parameters.

The goal in this example is to derive the SS_{comb} of the local buffer's read port, shared by W/I/O-Reg (non-double-buffered). Fig. 4(a) to (b) shows Step 1 "Divide" that divides the shared memory port by operand and analyzes each operand's DTL without interference; Fig. 4(b) to (c) shows Step 2 "Combine" that combines unit DTLs to deduce the SS_{comb} of the shared memory port, considering interference. Based on the mapping and RealBW (assume 1 data per cycle between LB and Reg in this example), the periodic data movement of each DTL is visualized in (b), from which the attributes for each DTL are derived in (c). Finally, SS_{comb} is calculated based on Eq. (1).

D. Step 3: Integrate SS_{comb} across all memory levels to derive total temporal stall $SS_{overall}$

$SS_{overall}$ accounts for the parallel memory operation as well as multiple stall sources across all memory levels. For the memory operations that can be overlapped, $SS_{overall}$ takes the maximum of SS_{comb}, i.e., the shorter stall of one memory can be hidden under the longer stall of the other; otherwise, $SS_{overall}$ is the sum of all stalls, indicating one memory stall blocks the operation of other memories, regardless of whether the data in other memories are ready. Users can customize this memory parallel operation constraint based on the design.

If calculated $SS_{overall} \leq 0$, we take zero as its final value since no temporal stall; otherwise, $SS_{overall} > 0$ indicates temporal stall exist during computation.

E. System's Overall Latency and MAC Array Utilization

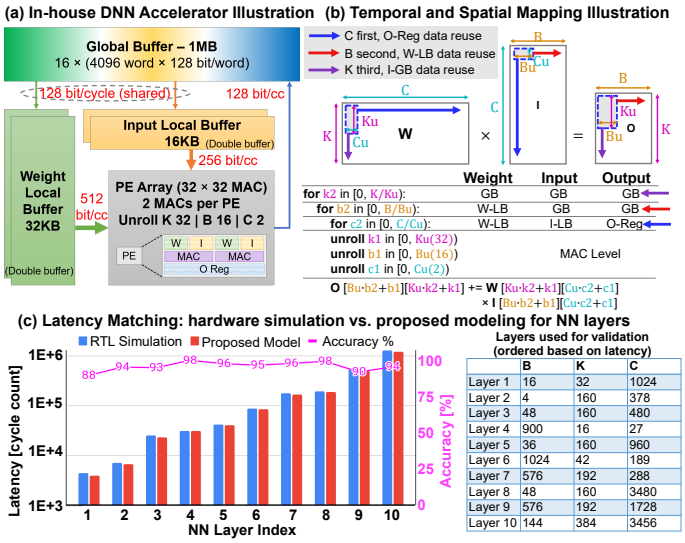
Based on the 3 Steps and data loading analysis, the system's overall latency (CC) can be derived by summing up the ideal computation cycles (CC_{ideal}), data loading cycles, spatial stall and temporal stall ($SS_{overall}$) (refer to Fig. 1). The overall MAC array utilization (U) can be deduced by CC_{ideal}/CC.

This uniform latency modeling enables DSE for various memory systems and mappings. It can also provide insights on identifying performance bottlenecks and optimization opportunities, as demonstrated in Section V.

IV. Validation

We validate the proposed latency model using an in-house DNN accelerator implemented in TSMC 7nm technology [18], designed for INT8-based inference tasks. For convolution layers, Im2Col operation (unrolling convolution into matrix-matrix-multiplication) is performed by a RISC-V core before processing on the accelerator. As shown in Fig. 5(a), this accelerator employs a systolic array-based design with 1K MAC units in a 16×32 PE array (2 MACs per PE) and one 24b Output register per PE. Each MAC connects to one 8b Weight and one 8b Input register. A total of 32KB and 64KB local buffer (LB) with 256b and 512b bus connection to PE array are used for temporal storage of Weight and Input, respectively. 1 MB global buffer (GB) tiled with 16 64KB SRAM macros is used. The mapping schemes are shown in Fig. 5(b).

We feed the latency model with the above hardware configuration. Fig. 5(c) shows the comparison of the modeled results with the hardware simulation, running NN layers (with different parameter sizes) of a hand-tracking workload [19]. For all evaluated NN layers, an average of 94.3% latency estimation accuracy is achieved.



	B	K	C
Layer 1	16	32	1024
Layer 2	4	160	378
Layer 3	48	160	480
Layer 4	900	16	27
Layer 5	36	160	960
Layer 6	1024	42	189
Layer 7	576	192	288
Layer 8	48	160	3480
Layer 9	576	192	1728
Layer 10	144	384	3456

Fig. 5. (a) Block diagram of the in-house DNN accelerator. (b) Temporal mapping and spatial unrolling schemes after Im2Col. (c) Model validation against the hardware RTL simulation running NN layers of different sizes.

V. Case Studies

In this section, we present 3 case studies to demonstrate how the enhanced latency model can be used to optimize the AHM design space. We integrate our model with ZigZag [8], a DNN accelerator architecture-and-mapping DSE framework, to generate various design points. For Case 1 and 2, the hardware architecture is fixed to a scale-down version of the in-house accelerator with 8×16 PE (2 MACs per PE, i.e. 16x16 MAC), 16KB Weight local buffer (W-LB), 8KB Input local buffer (I-LB), 1MB global buffer (GB) with 128 bit/cycle read/write BW, and a loop spatial unrolling of K 16 | B 8 | C 2. For Case 3, we perform architecture DSE by varying the hardware parameters. Im2Col layer transfer is applied to all the case studies.

A. Case 1: Mapping v.s. Latency

Different mappings lead to distinct latencies for the same DNN layer processed on the same hardware. Fig. 6 compares two different temporal mapping schemes: Mapping A and Mapping B, out of 30240 valid mappings obtained by ZigZag mapper. As shown in Fig. 6(c)(d), both mappings result identical ideal latency (CC_{ideal}) of 38400 clock cycles (cc), where Mapping A has 5% energy savings over Mapping B. Hence without considering temporal stall ($SS_{overall}$), Mapping A would be preferred. However, our latency model indicates that Mapping B actually has a 30% lower latency and 26% better MAC utilization (both spatial and temporal utilization included) over Mapping A, owning to its lower $SS_{overall}$.

The main difference between Mapping A and B is whether the C loop is split, highlighted by the blue boxes in Fig. 6(a)(b). This leads to different data reuse tradeoff between I and O at I-LB and GB levels: Mapping B adopts a full output stationary dataflow at O-Reg level (i.e., only final outputs write to GB) by scheduling all O's data reuse loops (C loops) at O-Reg level. On contrast, Mapping A has all I's data reuse loops (K loops) at I-LB level to reduce Input's data movement from GB to I-LB, at the cost of pushing part of O's data reuse loops (C loops) to the GB level (i.e., besides final Outputs, Partial Sums also need to be transferred between O-Reg and GB), as shown in Fig. 6(e). Note that W's data reuse distribution across memory levels in these two mappings are the same.

These differences in I and O data transfer cause different temporal stalls due to the insufficient GB BW compared to the required ones, shown in Fig. 6(f). Although Mapping A and B both exceed hardware's $Real_{BW}$ for GB write (e.g., 3072 vs. 128 bit/cycle), the stall is less for Mapping B since its GB write is less frequent (Fig. 6(e)). In addition, the Partial Sum transfer in Mapping A requires much higher GB read BW that cannot be met by the hardware's $Real_{BW}$. Thus, our memory-BW-aware latency model reveals the high $SS_{overall}$ in Mapping A that deteriorates MAC array utilization and system performance.

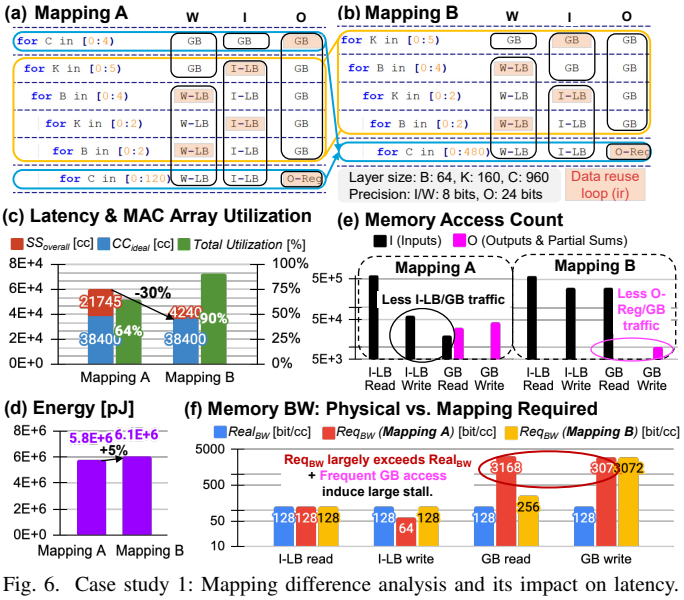


Fig. 6. Case study 1: Mapping difference analysis and its impact on latency.

This analysis clearly shows that a good latency model is instrumental to help DNN accelerator DSE mapper minimize $SS_{overall}$ by 1) matching Req_{BW} (mapping-dependent) with $Real_{BW}$ (HW-dependent), or 2) if $Real_{BW}$ is too low to match, reducing the frequent access of the low-BW link (e.g., reducing partial sum transfer in this case study).

B. Case 2: Workload Size v.s. Latency

DNN layer parameters largely impact execution time. In this case study, we use the same hardware parameters as in Case 1 and analyze the latency impact of the layer attributes (e.g., # of total MAC operation, operand size). Fig. 7(a) shows operand's percentage (W/I/O) and total MAC operation count by varying DNN layer dimensions from 8 to 512 for B/K/C. Fig. 7(b) shows the corresponding modeled Real latency and latency breakdown in terms of data pre-loading, ideal compute cycle, spatial stall, and temporal stall ($SS_{overall}$) as defined in Fig. 1.

Comparing Fig. 7(a) and (b), the Ideal latency matches with Total MAC Ops, where the Real latency follows the Total data size. The former is intuitive since it assumes 100% MAC array utilization with zero stall, where latter reveals the data movement bottleneck. Since the existing hardware has limited BW for GB (Fig. 6(f)), a fully output stationary dataflow at O-Reg level was always selected to minimize stall by reducing the GB access. When the layer is Output-dominant (large B and K, fewer C), its total data size increases compared to other layers under the same total MAC Ops due to the 24-bit O precision (v.s. 8-bit I and W), while at the same time the lower input channel count C, results in less output stationarity. This causes increased pressure on the GB write BW, causing the Real latency to deviate much more from the Ideal latency. For larger layer sizes (large C), Ideal computation cycle (green bars) dominate, and the deviation between Ideal latency and Real latency reduces.

Note that without including temporal stalls (i.e., the cyan dotted line), large discrepancies in latency estimation (e.g., 7.4× for layer (128,128,8) and 9.2× for layer (512,512,8)) occur (Fig. 7(b)), especially for layers with fewer input channels C.

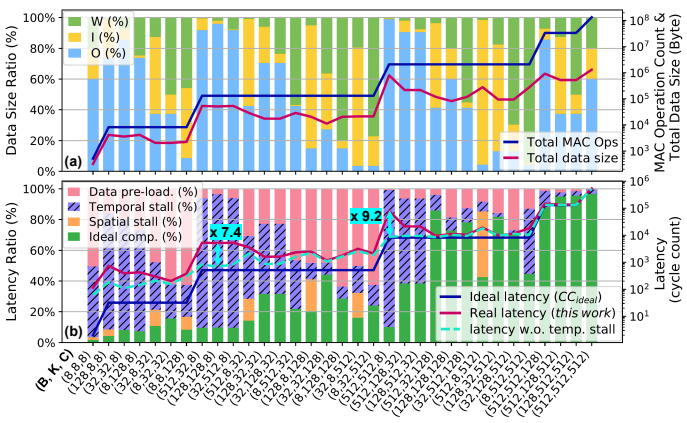


Fig. 7. Case study 2: Workload's impact on latency and latency breakdown.

C. Case 3: Hardware Architecture Design Space v.s. Latency

Previous case studies have shown the importance of the proposed latency model for evaluating the mapping and algorithm impact on a fixed hardware architecture. In this case study, we further take advantage of the model's generality and memory-BW-awareness to assess the impact of different HW architecture parameters (e.g., MAC array size, memory capacity and memory BW), and showcase how the design space changes with the presence of temporal stalls ($SS_{overall}$). We choose the following the MAC array sizes and scale the spatial mapping accordingly: 16×16 (spatial mapping as K 16 | B 8 | C 2), 32×32 (K 32 | B 16 | C 2), 64×64 (K 64 | B 32 | C 2). We construct a memory pool containing tens of register/memory candidates with different capacities to replace the W-/I-/O-Reg, W-/I-LB in the design space search. The GB size is 1MB for all the cases, where GB BW varies from 128 to 1024 bit/cycle. The area of GB is not included in the comparison.

Fig. 8 illustrates the latency-area design space for 4,176 hardware designs. Different MAC array sizes are shown in different colors, while dots that share the same color vary in memory hierarchy. For each design point, mapping optimization for lowest latency is performed.

Fig. 8(a) first shows the results using a memory-BW-unaware latency model. Since memory BW impact induced $SS_{overall}$ is ignored, all the architectures with the same array size achieve similar latency. Hence the minimum area design (i.e., with less memory) could be considered as optimal (close to the preferred corner), since larger memory capacity does not offer latency benefits but add area cost.

However, the conclusion changes once the memory BW impact is included. Fig. 8(b) and (c) show the design spaces obtained with our proposed model for GB BW of 128 bit/cycle (low BW) and 1024 bit/cycle (high BW), respectively, with the optimal design points highlighted. For both high and low GB BWs, different memory size combinations at Register and Local Buffer levels can impact the area-latency trade-off for a fixed MAC array size (i.e., same theoretical peak performance). For example, the 16×16, 32×32 and 64×64 array achieve their own lowest latency with moderate memory area cost at 128 bit/cycle GB BW. Only when the GB BW is high, the design points of the same array size cluster around the similar latency, indicating less latency impact from improving local memory storage and data reuse. This reveals the impact of $SS_{overall}$ in BW-limited systems, where the memory hierarchy needs to be optimized to maximize data reuse below that bottleneck memory level for stall reduction.

Another observation from the memory-BW-aware latency modeling is that the MAC array size preference can change for different memory BWs (Fig. 8 (b)(c)). At low GB BW, the optimal latency of 32×32 array can outperform that of the 64×64 array. Only at high GB BW, the 64×64 array can further improve the latency Pareto front.

In summary, BW-awareness is important for hardware design parameter optimization on latency. Recent technology advancement such as 3D IC technology with fine-pitch SRAM-on-logic stacking can offer energy-efficient high BW interconnects (e.g., >1024 bit/cycle) over the conventional 2D design. The proposed BW-aware latency model can aid in evaluating the impact of this new technology on the design space.

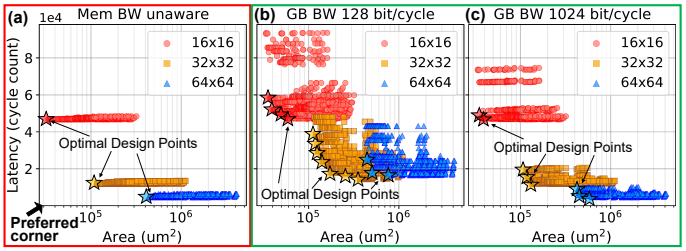


Fig. 8. Case study 3: Hardware architecture's impact on latency-area tradeoff.

VI. Conclusion

A unified analytical intra-layer latency model for DNN accelerators is proposed to support diverse architectures and mappings. Following a 3-step approach, our model overcomes prior challenges by systematically estimating the system's temporal stalls, capture the periodic operation of hardware components, and identify performance bottlenecks. This model is verified with an in-house DNN accelerator on various DNN layers with different mappings, achieving >94.3% accuracy on average. Three case studies from a mapping, workload and hardware perspective reveal the advantages of using the proposed model for DSE, as well as the importance of temporal stall modeling in exploring co-design opportunities for latency optimization. This intra-layer latency model builds a solid foundation for future work of modeling and optimizing latency in cross-layer multi-core DNN mapping scenarios.

References

[1] M. Capra *et al.*, "An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks," *Future Internet*, 2020.

[2] Y.-H. Chen *et al.*, "Using dataflow to optimize energy efficiency of deep neural network accelerators," *IEEE MICRO*, 2017.

[3] T.-J. Yang *et al.*, "A method to estimate the energy consumption of deep neural networks," in *ACSSC*, 2017.

[4] A. Parashar *et al.*, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *IEEE ISPASS*, 2019.

[5] Y. N. Wu *et al.*, "Accelergy: An architecture-level energy estimation methodology for accelerator designs," in *IEEE/ACM ICCAD*, 2019.

[6] H. Kwon *et al.*, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings," *MICRO*, 2020.

[7] X. Yang *et al.*, "Interstellar: Using halide's scheduling language to analyze dnn accelerators," *ASPLOS*, 2020.

[8] L. Mei *et al.*, "ZigZag: Enlarging Joint Architecture-Mapping Design Space Exploration for DNN Accelerators," *IEEE TC*, 2021.

[9] T. Tang *et al.*, "NeuroMeter: An integrated power, area, and timing modeling framework for machine learning accelerators industry track paper," in *IEEE HPCA*, 2021.

[10] E. Cai, D.-C. Juan *et al.*, "NeuralPower: Predict and deploy energy-efficient convolutional neural networks," *ArXiv*, 2017.

[11] T. Chen *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *OSDI*, 2018.

[12] R. Venkatesan *et al.*, "MAGNet: A modular accelerator generator for neural networks," in *IEEE/ACM ICCAD*, 2019.

[13] A. Samajdar *et al.*, "A systematic methodology for characterizing scalability of dnn accelerators using SCALE-Sim," in *IEEE ISPASS*, 2020.

[14] F. Munoz-Martinez *et al.*, "STONNE: Enabling cycle-level microarchitectural simulation for dnn inference accelerators," *IEEE CAL*, 2021.

[15] S. Dai *et al.*, "Fast and accurate estimation of quality of results in high-level synthesis with machine learning," in *IEEE FCCM*, 2018.

[16] M. Phothilimthana *et al.*, "Learned TPU cost model for XLA tensor programs," in *Workshop on ML for Systems at NeurIPS*, 2019.

[17] Y. Zhao *et al.*, "DNN-Chip predictor: An analytical performance predictor for dnn accelerators with various dataflows and hardware architectures," in *IEEE ICASSP*, 2020.

[18] S.-Y. Wu *et al.*, "A 7nm cmos platform technology featuring 4th generation finfet transistors with a 0.027um2 high density 6-t sram cell for mobile soc applications," in *IEDM*, 2016.

[19] D. Victor, "Handtrack: A library for prototyping real-time hand trackinginterfaces using convolutional neural networks," *GitHub repository*, 2017. [Online]. Available: https://github.com/victordibia/handtracking/tree/master/docs/handtrack.pdf