FTDL: A Tailored FPGA-Overlay for Deep Learning with High Scalability

Runbin Shi[1,†], Yuhao Ding[1,†], Xuechao Wei[2], He Li[3,*], Hang Liu[4], Hayden K.-H. So[1,*], Caiwen Ding[5]
[1]The University of Hong Kong, [2]Peking University, [3]University of Cambridge, [4]Stevens Institute of Technology, [5]University of Connecticut
{rbshi, yhding, hso}@eee.hku.hk, xuechao.wei@pku.edu.cn, he.li.grus@gmail.com, hliu77@stevens.edu, caiwen.ding@uconn.edu

***Abstract*—Fast inference is of paramount value to a wide range of deep learning applications. This work presents FTDL, a highly-scalable FPGA overlay framework for deep learning applications, to address the architecture and hardware mismatch faced by traditional efforts. The FTDL overlay is specifically optimized for the tiled structure of FPGAs, thereby achieving post-place-and-route operating frequencies exceeding 88 % of the theoretical maximum across different devices and design scales. A flexible compilation framework efficiently schedules matrix multiply and convolution operations of large neural network inference on the overlay and achieved over 80 % hardware efficiency on average. Taking advantage of both high operating frequency and hardware efficiency, FTDL achieves 402.6 and 151.2 FPS with GoogLeNet and ResNet50 on ImageNet, respectively, while operating at a power efficiency of 27.6 GOPS/W, making it up to $7.7\times$ higher performance and $1.9\times$ more power-efficient than the state-of-the-art.**

I. Introduction

There is an increasing demand to conduct the inference of deep learning (DL) *faster* [1]–[9]. Of mainstream hardware accelerators, field-programmable gate arrays (FPGAs), offering both massive parallelism and high energy efficiency, are hence the ideal platform to expedite inference. However, most existing works simply deploy the application-specific integrated circuit (ASIC)-oriented architectures to FPGA without considering the FPGA underlying layout, which leads to the *architecture-layout mismatch*. Such a mismatch subsequently introduces irregular routing across underlying FPGA hardware components, which substantially hampers the timing performance when scaling this design up. For instance, the widely adopted systolic array liked architecture, which connects data memories to the *boundary* processing element (PE) in the PE array, suffers from this mismatch in FPGA implementation. Because the block RAMs (BRAMs, memory component in FPGA) are uniformly distributed on the FPGA chip, simply connecting massive BRAMs to a boundary PE leads to irregular routing distances, and the resultant timing reduction harms the computational throughput in proportional. Besides the aforementioned hardware ***scalability*** issue, the existing DL architecture also experiences the underutilization problems during scale-up. ***Hardware efficiency***, the ratio of attainable computational throughput to the theoretical value, is leveraged to measure the computational resource utilization. Most existing works achieved a hardware efficiency of 45-70% [4], [5], [8], [10]. Although increasing batch size can maintain high hardware efficiency [9], it is infeasible for edge devices that need low latency.

FTDL (FPGA tailored deep learning) addresses the scalability and hardware efficiency issues via an FPGA layout-aware architecture along with optimal scheduling scheme of compiler. The contributions of FTDL can be summarized as,

- FTDL proposes a novel *overlay architecture* for convolutional and fully connected layers. This overlay is tailored for the tiled structure of modern FPGAs, allowing post-place-and-route operating frequencies to reach over 88 % of the theoretical DSP operating frequency across different devices and design scales.
- FTDL provides a compilation tool that maps most DL layers to the overlay with over 80 % hardware efficiency on average. Along with the high operating frequency, FTDL achieves 402.6 FPS and 151.2 FPS on ImageNet with GoogLeNet and ResNet50, respectively. We achieve at least $2.3\times$ and up to $7.7\times$ in performance compared to the prior arts. The power efficiency of FTDL is 27.6 GOPS/W, which is up to $1.9\times$ compared to previous designs.

TABLE I
MLPerf benchmarks for DL systems [11] (16-bit weight).

DL Model Structures	Applications	Operations			#Weight
		CONV	MM	EWOP	(bytes)
GoogLeNet	Image Processing	99.73%	0.07%	0.20%	13.7M
ResNet50	Image Processing	99.67%	0.05%	0.27%	51M
AlphaGoZero	Operation Decision	99.86%	0.08%	0.06%	2.08M
Sentimental-seqCNN	Sequence Analysis	89.86%	0.15%	9.99%	345.06K
Sentimental-seqLSTM	Sequence Analysis	0.00%	99.89%	0.11%	39.9M

II. Background

A. Generality of DL Workloads

In general, the computations in deep learning systems can be partitioned into three categories of sub-workloads, matrix multiply (MM, e.g., fully connected layer), convolution (CONV) and element-wise operation (EWOP, e.g., activation), where each sub-workload uses different hardware building blocks to speed up the execution. We studied the models in MLPerf benchmark set [11], as Table I, and found that CONV and MM account for over 90 % computational workload. Therefore, FTDL targets accelerating CONV and MM computation, while the EWOP is processed by host CPU in a pipeline fashion.

B. FPGA Friendly Design Morphology of DL Architecture

1) Storing Weights On-chip: Weight stationary is widely adopted in DL *inference* accelerator that maintains the weight in on-chip memory during runtime to save the bandwidth and energy [12]. Comparing to other devices, *FPGA* is suitable for the weight stationary scheme as its massive on-chip memory supplies abundant storage and bandwidth. As listed in Table I, with the quantization technique [13], the model weights are quantized to 16-bit fixed-point data, and the storage requirement is *in the same order of magnitude* (mega byte) to the on-chip memory resource of FPGA device. For cases that the model cannot reside in one chip, multi-FPGAs system [14] enables partitioning the model to multiple devices, which makes the weight stationary feasible to most application scenarios. Therefore, FTDL adopts the weight stationary scheme and provides a generic DL architecture that applies to both single- and multi-FPGA cases.

[†]Equal contribution.
[*]Corresponding authors.

2) Design with High Operating Frequency: For FPGA-based DL architecture, the attainable *maximum operating frequency* (f_{max}) is a critical factor that directly decides the theoretical *computation-throughput*. Most prior designs achieved a f_{max} below 250 MHz, while the theoretical f_{max} of individual digital signal processor (DSP) and look-up table (LUT) on FPGA are 740 MHz [15]. The 66% loss on f_{max} is sourced from the mismatch between the *irregular architecture design* and the *regular components layout of FPGA* (***architecture-layout mismatch***). Recently, researchers have proposed high-f_{max} designs dedicated for FPGA [16]–[18]. In the work of Samajdar *et al.* [16], the cascaded interconnections are leveraged to construct a design running at 650 MHz. However, the FPGA should be reconfigured for each layer as the design is tailored for a specific kernel size. The ***characteristics*** of designs with high f_{max} and high scalability are highlighted: First, fully utilize the f_{max} of DSP and LUT; Second, consume the resource in equilibrium; Third, the placement and routing of the design should be predictable (architecture-layout match). These characteristics are the principles and advantages of the FTDL hardware in this paper.

III. Distributed Architecture of FTDL

The FTDL is a *distributed* and *hierarchical* architecture. The distributed morphology fits FPGA well and benefits timing performance of the design; The hierarchical levels provide flexibility for the design scale-up and workload scheduling that facilitates the *hardware efficiency* for different DL workloads. The hardware composition of FTDL is described hierarchically in the following sub-sections.

A. Tiled-PE: The Basic Computational Unit

1) Layout-aware PE design: The tiled processing element (TPE), as Fig. 1, is the basic computational unit that includes a digital signal processor (DSP), configurable logic blocks (CLBs), and block RAM (BRAM). The DSP performs multiply-accumulate (MACC) operation, the most intensive computation operation in both MM and CONV workloads. The BRAM stores a portion of trained model weights that are required in the workload of this TPE. With the weight-stationary scheme, the weights are preloaded to BRAM during FPGA initialization. The distributed RAM (constructed by CLBs) stores input-activation. In particular, the numbers of BRAM/CLB spent on the weight buffer (WBUF) and activation buffer (ActBUF) in TPE are configurable, and the values are determined by the resource amount in the target FPGA device. For instance, in the layout of FPGA primitive (underlying component) as Fig. 1, the resource ratio of DSP to BRAM18 is one (a common ratio in most devices). Thus, each TPE contains one DSP and one BRAM18 primitive proportionally. The mapping process in FPGA implementation automatically groups and places the primitives (including a DSP, a BRAM18, and several CLBs) of a TPE to a local fabric area. The design advantages are in two-fold: (i) The wires inside a TPE connect the primitives in a local fabric area, that leads to an ultra-low net delay and a high f_{max}; (ii) During connecting multiple TPEs, the intrinsic *cascade interconnection* in the DSP column can be used for accumulation of MACC operation. The interconnection can achieve the same f_{max} as the DSP and avoid the cost of extra routing resources.

2) Double-pump for optimal f_{max}: According to vendor statistics [15], DSP and CLB are capable of operating at a f_{max} close to 740 MHz, while the theoretical f_{max} of BRAM is ≈528 MHz. We adopt *double-pump* technique [19] to operate different primitives at their theoretical f_{max}. As shown in Fig. 1, BRAM is driven by a low-speed clock $\mathtt{CLK_l}$, while the DSP and distributed RAM are driven by a synchronized $\mathtt{CLK_h}$ with double frequency of $\mathtt{CLK_l}$. The *double-pump*

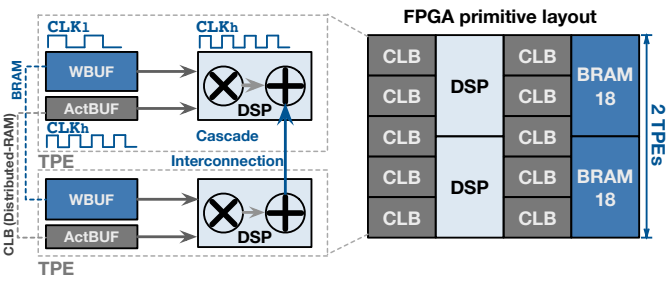

Fig. 1. The tiled processing element (TPE) in FTDL, an FPGA layout friendly design. The intrinsic DSP-interconnections accumulate the results of TPEs.

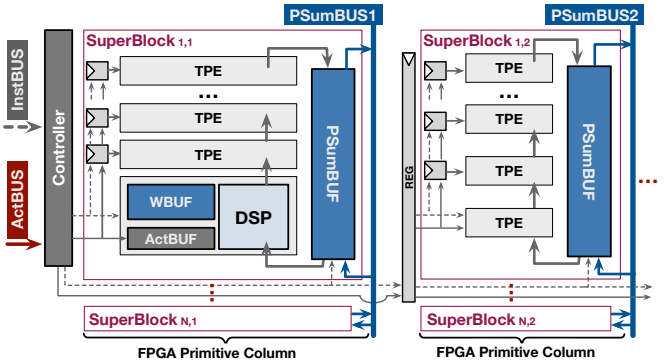

Fig. 2. The SuperBlock organizes multiple TPEs in one column. Each SuperBlock contains a partial sum buffer (PSumBUF) for accumulation.

design orchestrates the workloads of FTDL (MM and CONV) since each weight is usually reused for multiple times during computation. Specifically, the DSP uses different activation values (from distributed RAM) but same weight (from BRAM) in each two consecutive clock cycles $\mathtt{CLK_h}$, in order to achieve maximal computational throughput for the whole system.

B. SuperBlock: Proper Organization of TPEs

As a primitive column in FPGA can have up to 240 DSPs that compose the same number of TPEs. Simply cascading all TPEs in one column loses the control-flexibility. As depicted in Fig. 2, FTDL partitions the TPEs in one column and groups them to SuperBlocks, and each SuperBlock is directed by an independent Controller. Furthermore, one SuperBlock contains an on-chip partial-sum buffer (PSumBUF) that stores intermediate results and therefore saves the off-chip memory bandwidth. As Fig. 2 shows, the first and the last TPE inside each SuperBlock are connected to the partial sum buffer (PSumBUF), which stores and fetches the results for accumulation of MM or CONV periodically. Thus, in temporal order, the TPE can operate on the computations that sum up to different addresses in PSumBUF. In the run-time, the Controller decodes the config from instruction bus (InstBUS) and generates proper memory control signals to ActBUF, WBUF, and PSumBUF that collaborate on the computation or data transmission. Besides, the PSumBUS is connected to SuperBlock that stores the results in PSumBUF to the off-chip memory; In case the final results cannot be obtained in one pass of on-chip computation, the PSumBUS can also store and reload the intermediate results for multi-pass computation.

C. *Scale-up* FTDL *Hardware*

Scalability is the competitive advantage of FTDL that facilitates the users to deploy it on most FPGA devices while maintaining a high f_{max}. This attribute takes advantage of the FTDL hardware scale-up method as follows.

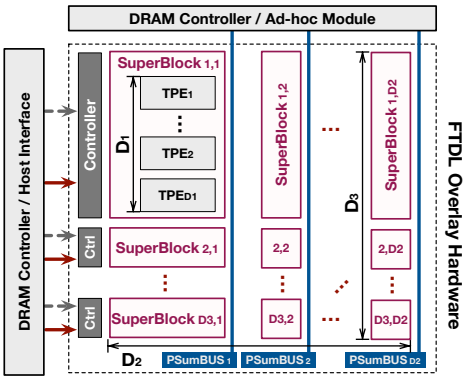

Fig. 3. The general system diagram with parameterized FTDL hardware.

In the horizontal axis, as Fig. 2, multiple `SuperBlocks` are organized with the scheme of single instruction, multiple data (SIMD). The `Controller` locates at the left side of a `SuperBlock` row, and the control signals are bypassed to all the `SuperBlocks` in the same row via pipeline registers. Furthermore, the input activation data (from the `ActBUS`) is also shared among the horizontal `SuperBlocks`. Thus, the `SuperBlocks` in one row perform synchronized operations with the same *activation* data but different *weights* in `WBUF`, which are preloaded during FPGA configuration. We adopt SIMD and input activation sharing scheme since the CONV and MM workload both have regular computation patterns and activation reusability.

In the vertical axis, as Fig. 2, multiple `SuperBlock` rows are stacked and working independently. Each `SuperBlock` row owns a `Controller`, `ActBUS`, and `InstBUS`, while the `SuperBlock` column shares the same `PSumBUS`. The data stream on `PSumBUS` can be statically organized to avoid congestion.

D. *Embedding* FTDL *To Your Design*

FTDL can be embedded in most FPGA-based DL-system. Fig. 3 shows the generic system diagram while the details vary from case to case. The `InstBUS` and `ActBUS` are connected to DRAM controller or host interface to receive the input activation or instruction stream. Besides the DRAM controller, `PSumBUS` can also be connected to other hardware with a streaming manner, such as an ad hoc pooling module or activation module that is designed for a particular application.

Importantly, the FTDL hardware is parametrizable as labeled on Fig. 3, where D_1 is the `TPE` number in each `SuperBlock`, D_2, and D_3 are the numbers of `SuperBlock` column and row, respectively. To make the 2D hardware suitable for FPGA layout, we set constraints on the parameter values. First, D_2 is less than the number of DSP column on the FPGA device; Second, $D_1 \times D_3$ should be less than the number of DSP in each column, that ranges from 20 to 240 in different FPGA devices. As the values of D_1 and D_3 affects the overall hardware cost and efficiency, the FTDL compiler in Section IV can automatically select proper values for user workloads.

E. *Periodic Control flow with Double-buffering*

This section elaborates on the control flow of FTDL, that is corresponding to the `Controller` signals generated for each component.

As the content in `ActBUF` and `PSumBUF` are updated during execution, the control flow consists of *computation* and *communication* two portions. To execute these two portions concurrently, double-buffering technique is applied to `ActBUF` and `PSumBUF`, where each buffer is divided into two *sub-buffers* logically. In the run-time, one sub-buffer provides data for computation, and the rest one is updated (load/store) with off-chip memory. The two sub-buffers exchange the

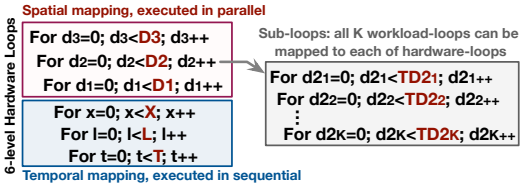

Fig. 4. The tiled loops represent the workload scheduling in spatial and temporal; The trips counts of sub-loops compose the mapping vector. Note that both CONV and MM are analyzed as a K-level nested loop.

role (computation or communication) periodically. List 1 illustrates the hardware behavior for a workload that contains $X \times L \times T$ MACC operations. Note that the original 1-level loop with a trip count of $X \times L \times T$ is *tiled* to 3-level nested loops (`LoopX`, `L`, `T`). This is because the storage of on-chip buffers are limited and usually can fit in neither the entire activation data nor the partial sum. Furthermore, the storage of `ActBUF` (distributed-RAM, 64-256 words) comes much smaller than that of `PSumBUF` (BRAM, 1024-4096 words). Thus, the `LoopT` is set for `ActBUF`, which is updated every T cycles; and `LoopL` is set for `PSumBUF` that communicates with off-chip memory every $T \times L$ cycles. For real DL application, the T and L vary according to *computational intensity*, which is decided by the workload and scheduling scheme. Section IV further elaborates workload scheduling on FTDL that facilitates high hardware efficiency.

IV. Workload Compilation for FTDL

The DL application consists of multiple layers with either MM or CONV computation, and the hyperparameter (the trip count of each loop) varies from layer to layer. Based on the hardware proposed above, FTDL also provides the compilation scheme that efficiently maps the custom CONV/MM layer to the parameterized hardware. The FTDL compiler generally addresses two problems,

- With a particular FTDL hardware configuration (fixed D_1, D_2, D_3 in Section III-D), how to schedule a DL layer to achieve the *minimum execution time*?
- With the same resource utilization ($D_1 \times D_2 \times D_3$ is a const), what is the best hardware configuration for a DL-layer?

A. *Abstraction Model of* FTDL *Scheduling*

To completely explore the workload scheduling space, the FTDL hardware is formally abstracted as the nested loops in Fig. 4, in which we use the *tiled loops* to represent the workload partition and mapping behavior. We use 6-level nested loops to represent the FTDL hardware behavior, where loop D_1, D_2, D_3 represents the spatial resources of `TPE`, `SuperBlock` column, and `SuperBlock` row, respectively. These loops are executed in parallel on the FTDL hardware. The loop levels X, L, T are corresponding to the `TPE` control flow demonstrated in List 1, that are executed in sequential. Besides these *hardware* loops, we represent the *workload* using a K-level nested loop. For instance, K is 3 in MM and 6 in CONV. Theoretically, each *workload loop* can be partitioned and mapped to any *hardware loop*. Therefore, as Fig. 4 shows, each hardware loop can be expanded to K sub-loops to represent the map information; The trip counts of sub-loops for each hardware level is denoted as $TD1_k$, $TD2_k$ ... TT_k, where $k \in [1, K]$. Each trip count represents the workload size that scheduled to a particular hardware level. For instance, $TD1_k$ is the tile size in the k_{th} workload loop that

```
1 LoopX: for (x=0; x<X; x++){
2  update_PSumBUF(); //update PSumBUF every L*T cycles
3  LoopL: for (l=0; l<L; l++){
4   update_ActBUF(); //load ActBUF every T cycles
5   LoopT: for (t=0; t<T; t++){
6    TPE_OP(x,l,t) }}}; //MACC operation on TPE every cycle
```

Listing 1. Pseudo code of control flow: the hardware behavior is represented by 3-level loops with hierarchical buffer update.

are spatially mapped to TPEs in one SuperBlock. With these trip counts, both the spatial and temporal mapping between workload and hardware are unique, and the relation can be formulated as,

$$\texttt{TPE}[d_3][d_2][d_1][x][l][t] \Leftarrow workload[i_1][i_2]...[i_K] \tag{1}$$

$$(i_1, ..., i_K)^T = [\mathbf{T} \cdot \mathbf{H}] \cdot (d3, d2, d1, x, l, t)^T \tag{2}$$

$$\mathbf{T} = (\overrightarrow{TD1}, \overrightarrow{TD2}, \overrightarrow{TD3}, \overrightarrow{TX}, \overrightarrow{TL}, \overrightarrow{TT})^T \tag{3}$$

$$= \begin{bmatrix} TD1_1 & TD2_1 & TD3_1 & TX_1 & TL_1 & TT_1 \\ TD1_2 & TD2_2 & TD3_2 & TX_2 & TL_2 & TT_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ TD1_K & TD2_K & TD3_K & TX_K & TL_K & TT_K \end{bmatrix} \tag{4}$$

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ D_2 & 1 & 0 & 0 & 0 & 0 \\ D_1 & D_1 & 1 & 0 & 0 & 0 \\ X & X & X & X & 1 & 0 \\ L & L & L & L & L & 1 \\ T & T & T & T & T & T \end{bmatrix} \tag{5}$$

where the (d_3, d_2, d_1) are the indices of spatial hardware loops, (x, l, t) are the indices of temporal hardware loops, $(i_1, ..., i_k)$ are the indices of the workload loops. The symbol $\Leftarrow$ in Eqn. 1 represents the workload mapping. The mapping relation is the product of matrix $\mathbf{T}$ and $\mathbf{H}$, where matrix $\mathbf{T}$ is composed of the ***mapping vectors*** (as Eqn. 3), each contains the trip counts of K sub-loops. Eqn. 5 shows $\mathbf{H}$, where D_1, D_2, D_3 is the hardware configuration, X, L, T can be calculated as Eqn. 6,

$$X = \prod_{k=1}^{K} TX_k, L = \prod_{k=1}^{K} TL_k, T = \prod_{k=1}^{K} TT_k, \tag{6}$$

Therefore, we formulate the scheduling problem as finding the proper *mapping vectors* $(\overrightarrow{TD1}, \overrightarrow{TD2}, \overrightarrow{TD3}, \overrightarrow{TX}, \overrightarrow{TL}, \overrightarrow{TT})$ that achieve the optimal system performance. The following contents elaborate on the analytical model, objectives, and constraints of this problem.

B. Analytical Model

With the candidature mapping vectors, the analytical model is given to fast evaluate the performance in three aspects: (i) computation time, (ii) communication time, (iii) WBUF utilization. The time consumption is measured with the cycle number of $\texttt{CLK}_\text{h}$.

1) Computation Time (C_{comp}): It is the time cost by MACC operation for the workload layer, regardless of the communication time. We have,

$$C_{comp} = \prod_{k=1}^{K} TX_k \times (\prod_{k=1}^{K} TL_k \times \prod_{k=1}^{K} TT_k + Lat) \tag{7}$$

where the pipeline latency (Lat) of TPE-chain inside a SuperBlock is considered, and we have $Lat = D_1 + 6$ in our implementation.

2) Communication Time (C_{comm}): In general, two types of data communication exist in FTDL system, the off-chip DRAM interface and on-chip interface, i.e., ActBUS and PSumBUS. Each interface bandwidth can be the *potential performance bottleneck*. We formulate the communication time over on-chip interfaces as,

$$C_{comm}^{\texttt{ActBUS}} = f_{act}(\overrightarrow{TT}) \times \prod_{k=1}^{K} TX_k \times \prod_{k=1}^{K} TL_k \tag{8}$$

$$C_{comm}^{\texttt{PSumBUS}} = f_{psum}(\overrightarrow{TT}, \overrightarrow{TL}) \times \prod_{k=1}^{K} TX_k \times D_3 \tag{9}$$

where $C_{comm}^{\texttt{ActBUS}}$ and $C_{comm}^{\texttt{PSumBUS}}$ are the cycle number for ActBUS and PSumBUS, respectively. As the data reusability is different for MM and CONV cases, we use function f_{act} and f_{psum} to represent the data tile size of activation and partial sum that loaded (stored) to (from) on-chip buffer in each round, respectively. For the DRAM interface, we defined $C_{comm}^{\texttt{DramRD}}$ and $C_{comm}^{\texttt{DramWR}}$ similarly to modeling the DRAM read or write time, that are obtained via dividing the transmission volume to the pre-set DRAM bandwidth ($B_{\texttt{DramWR}}$, $B_{\texttt{DramRD}}$).

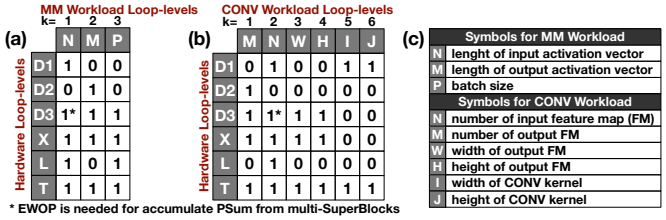

Fig. 5. Adjacency matrix in mapping for the MM (a) and CONV (b) workload.

3) WBUF efficiency ($E_{\texttt{WBUF}}$): As the weight in individual WBUF cannot be shared inter TPEs, *duplicated weight storage* exists in cases where two operations use the same weight value but are scheduled to different TPEs. However, less weight duplication means more workload layers can be arranged on one FPGA device. We defined WBUF efficiency ($E_{\texttt{WBUF}}$), the ratio of theoretical weight storage (without duplication) to practical value, to evaluate each mapping vector. $E_{\texttt{WBUF}}$ is formulated as a function with input of temporal mapping vectors ($\overrightarrow{TX}$,$\overrightarrow{TL}$,$\overrightarrow{TT}$).

C. Solution Space of Mapping Vectors

The mapping vectors contain $6 \times K$ elements that result in a huge exploration space in the optimal solution searching. This section defines the constraints that narrow down the set of candidature solutions.

1) Adjacency Matrix for Workload Mapping: With the general model in Section IV-A, each of K workload loops can be partitioned and mapped to an arbitrary level in hardware loops (6-levels). However, due to the hardware constraints, some loop level in CONV or MM cannot be mapped to a particular hardware loop, and therefore the corresponding element in mapping vectors should be 1. For instance, the workload mapped to D_2 loop of hardware should use exactly the same input activation, as the data on ActBUS are shared with SuperBlocks in one row; Thus, in the MM workload, only the M-loop can be mapped to hardware D_2-loop. A similar constraint exists in D_1-loop, as the results from D_1 TPEs are compulsorily accumulated in the SuperBlock. We formalize these constraints on mapping vectors with a $6 \times K$ *adjacency matrix* $\mathbf{A}$. The element $\mathbf{A}(i, j)$ is 0 or 1 to represent whether j-level of workload loop can be mapped to i-level of hardware loop. Once the element is 0, the corresponding value in mapping vector is set to const 1 during searching. The adjacency matrix for MM and CONV are given in Fig. 5(a)(b) respectively.

2) Logical Constraints: Three logical constraints exist on the mapping vectors. First, as Eqn. 10, the element products of the spatial mapping vectors $(\overrightarrow{TD1}, \overrightarrow{TD2}, \overrightarrow{TD3})$ should not exceed resource amounts D_1, D_2, D_3, respectively. Second, as Eqn. 11, for any $k \in [1, K]$, the product of the k_{th} elements of all mapping vectors should be greater or equal to the trip count (denoted as W_k) of the corresponding workload loop. This item ensures all the workloads are accomplished, even if the invalid computation exists.

$$\prod_{k=1}^{K} TD1_k \leq D_1, \prod_{k=1}^{K} TD2_k \leq D_2, \prod_{k=1}^{K} TD3_k \leq D_3, \tag{10}$$

$$\forall k \in [1, K], TD1_k \times TD2_k \times TD3_k \times TX_k \times TL_k \times TT_k \geq W_k \tag{11}$$

Third, with a particular mapping vector, the on-chip memory requirements should below the memory capacity in hardware configuration. FTDL modeling the buffer utilization function for CONV/MM workload, and denote the designate capacity of each buffer as $S_{\texttt{ActBUF}}$, $S_{\texttt{WBUF}}$, $S_{\texttt{PSumBUF}}$. The detail is emitted due to space limitation.

D. Putting It All Together

Combining the aforementioned *mapping abstraction, analytical model and constraints*, FTDL finalizes the objectives of the scheduling problem as follows,

1) Objective1: optimal performance: Given the hardware configuration (D_1, D_2, D_3, $S_{\texttt{ActBUF}}$, $S_{\texttt{WBUF}}$, $S_{\texttt{PSumBUF}}$, $B_{\texttt{DramRD}}$, $B_{\texttt{DramWR}}$), find the optimal mapping vectors ($\overrightarrow{TD1}$, $\overrightarrow{TD2}$, $\overrightarrow{TD3}$, $\overrightarrow{TX}$, $\overrightarrow{TL}$, $\overrightarrow{TT}$), that minimizes the workload execution time,

$$\min_{\mathbf{T}\in\mathcal{D}} C_{exe} = \max(C_{comp}, C_{comm}^{\texttt{PSumBUS}}, C_{comm}^{\texttt{ActBUS}}, C_{comm}^{\texttt{DramRD}}, C_{comm}^{\texttt{DramWR}}) \quad (12)$$

where $\mathcal{D}$ represents the feasible searching space defined in Section IV-C; C_{exe} is the overall execution time (in cycle), which is the maximum value of all performance metrics defined in Section IV-B.

2) Objective2: balance between performance and WBUF *efficiency:* In current, FTDL evaluates performance with a single layer workload. One FPGA device is supposed to store weight values of multiple layers and compute different layers in sequential. Thus, in these cases, the WBUF efficiency ($E_{\texttt{WBUF}}$) should be considered along with the execution time in evaluation. We define a *score* to indicate the balance between performance and $E_{\texttt{WBUF}}$. As,

$$\max_{\mathbf{T}\in\mathcal{D}} Score = C_{exe}/C_{exe}^{min} + E_{\texttt{WBUF}} \quad (13)$$

where C_{exe}^{min} is the theoretical minimum execution time with particular hardware. By doing so, two portions of Eqn. 13 are normalized, and the $Score$ is the summation of these two factors. The objective is to find optimal mapping vectors that maximizes the $Score$.

3) Objective3: optimal hardware configuration: Besides searching the optimal mapping vector with a particular hardware configuration, this objective is to find the optimal hardware configuration at the same cost. Given the number of TPE ($D_1 \times D_2 \times D_3$), FTDL generates feasible sets of (D_1, D_2, D_3) and replay the previous searching on each search. Then the best hardware configuration with corresponding optimal mapping is generated.

4) Searching Scheme: With the multiple constraints, the search space is the integer hull of a non-convex high dimensional polytope. Thus, it is infeasible to solve the problem via analytical method. In our searching scheme, all the candidature solutions are generated under the guidance of adjacency matrix. Subsequently, FTDL performs the constraints check over the candidates and excludes the infeasible solutions. Finally, FTDL traverses the feasible solutions and select top-k optimal mapping vectors as the final scheduling schemes.

V. Experiments and Evaluations

A. Experimental Methods

To evaluate the hardware and workload compilation proposed above, we implemented FTDL as a framework. The hardware part is written with parameterized Verilog RTL, verified with Synopsys VCS, and implemented by Vivado 2018.2. Particularly, the *primitive macros* of distributed RAM, BRAM, and DSP are leveraged to realize the fine-grained hardware design. This method improves the predictability of hardware implementation, as the synthesis tools map the primitives to underlying components directly instead of inferencing them to other logic. The hardware evaluation results are presented in Section V-B.

We also implemented the FTDL compiler that accepts input of the DL layer attributes (hyperparameters and layer type) and the hardware configurations(D_1, D_2, D_3, and buffer sizes). After searching the feasible space, the compiler gives the optimal mapping vectors for both Objective1, 2 in Section IV-D. The compiler also dumps the control instructions for all Controllers in target FTDL, that can be loaded via InstBUS before the layer execution. With the layers in MLPerf benchmarks (Table I), RTL simulation was conducted to obtain the hardware efficiency and real performance. The data access trace was dumped and sent to the DRAMPower [20], an accurate model that supplies the DRAM performance. The DRAM bandwidth is set to 26 GB/s that is achievable in most platforms. Note that the EWOP layers were allocated to host CPU, and the performance was not bounded by these layers. The overall performance and comparison are presented in Section V-C.

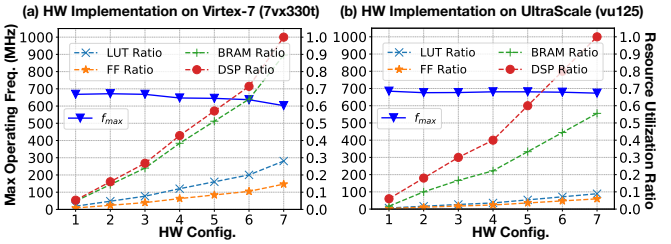

Fig. 6. Hardware evaluation on Xilinx-Virtex device (a) and UltraScale device (b) after place and route. Seven hardware configurations are evaluated in a scale-up fashion. The FTDL hardware design shows a good timing and scalability that f_{max} stabilizes above 620 MHz on Virtex and 650 MHz on UltraScale.

B. Hardware Evaluation

As the significant advantages of FTDL hardware, the timing and scalability are evaluated on different FPGA devices. Fig. 6 presents the hardware results reported by Vivado after place and route, where (a) is with a Virtex-7 device (7vx330t), and (b) is with an UltraScale device (vu125). For each device, seven hardware configurations were evaluated in the *scale-up* fashion. On both devices, the f_{max} of DSP ($\texttt{CLK}_\text{h}$) stabilizes above 620 MHz during the scale-up, even the BRAM and DSP are 100% utilized. The layout-aware design of FTDL contributes to the good timing and scalability.

C. Working with Real-world Applications

Besides the high scalability and good timing on the hardware aspect, FTDL compiler facilitates high hardware efficiency in mapping different DL layers. The execution performance of FTDL is evaluated with MLPerf benchmarks (GoogLeNet and ResNet50) on ImageNet dataset that the frame size is 224×224. FTDL hardware used an example configuration ($D_1 = 12$, $D_2 = 5$, $D_3 = 20$) on the UltraScale-vu125 device that has 1200 DSPs, corresponding to 1200 TPEs, and the DSP operation frequency is set to 650 MHz.

1) Performance Visualization: To comprehend the optimal mapping solution searching, FTDL provides a roofline visualization tool for performance analysis. Fig. 7 shows the roofline plot with solutions for a CONV layer, where the color represents the WBUF efficiency defined in Section IV-B. Two sub-figures (a)(b) plot *top-200* optimal solutions for objectives on *performance*(Obj.1) and *balance*(Obj.2), respectively, and the chart is zoomed into the area of interest. For Fig. 7(a), a near-to-roof performance is reached, while most solutions suffer a low WBUF efficiency ($E_{\texttt{WBUF}} \approx 0.2$). However, in Fig. 7(b), the top-200 *balanced* solutions with Obj.2 are plotted that all achieve a high $E_{\texttt{WBUF}}$ (≈ 1) with only a slight loss on the attainable performance. Apparently, for this example layer, the solutions with Obj.2 are preferable as they save WBUF $5\times$ than solutions with Obj.1. With the roofline-visualization tool, users can definitely conduct performance analysis in visual for their DL layers, and further select an optimal solution with the trade between performance and hardware cost.

2) Overall Performance and Related Works Comparison: With the FTDL framework, we obtained the optimal mapping solutions for each layer in GoogLeNet and ResNet50. The comparison with

TABLE II
OVERALL PERFORMANCE OF FTDL AND COMPARISON WITH RELATED WORKS.

Work	[10]	[2]	[3]	[4]	[5]	[7]	[8]	[21]	[1]	[9]	**FTDL (this work)**
Weight Quantization	16-bit	16-bit	16-bit	16-bit	16-bit	16-bit	16-bit	16-bit	16-bit	16-bit	**16-bit**
DSP Frequency (MHz)	150	100	125	167	200	200	150	150	170	240	**650**
Hardware Efficiency	45.4%	73.0%	72.0%	67.5%	48.3%	48.2%	71.9%	70.8%	76.5%	89.1%	**81.1%/74.8%**
GoogLeNet Perf. (FPS)	52.0 (1.0x)	55.7 (1.1x)	68.7 (1.3x)	86.1 (1.7x)	73.8 (1.4x)	73.5 (1.4x)	82.3 (1.6x)	81.1 (1.6x)	99.3 (1.9x)	163.3 (3.1x)	**402.6 (7.7x)**
ResNet50 Perf. (FPS)	21.2 (1.0x)	22.7 (1.1x)	28.0 (1.3x)	35.0 (1.7x)	30.1 (1.4x)	29.9 (1.4x)	33.5 (1.6x)	33.0 (1.6x)	40.4 (1.9x)	66.5 (3.1x)	**151.2 (7.1x)**
Power Efficiency (GOPS/W)	N/A	16.8 (1.2x)	N/A	21.4 (1.5x)	N/A	N/A	14.5 (1.0x)	30.4 (2.1x)	N/A	N/A	**27.6 (1.9x)**

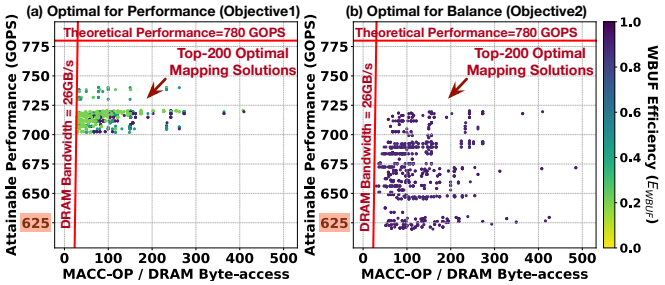

Fig. 7. Roofline-based visualization tool for performance analysis. (a) and (b) plot top-200 optimal solutions by FTDL compiler for *performance* and *balance* objectives respectively. The solution in (b) is preferable as they saves `WBUF` $5\times$ to (a) with only slight performance loss. Note that the y-axis has been scaled to the area of interest.

previous works in terms of performance and power efficiency are presented in Section II. FTDL achieved an average hardware efficiency of 81.1% and 74.8% in GoogLeNet and ResNet50, respectively. That can be translated to a processing throughput of 402.6 and 151.2 frames per second (FPS) on ImageNet dataset processing. We also compared the design operating frequency and hardware efficiency on the same type of DL benchmarks and using same weight quantization scheme. For a fair comparison, we obtained the FPS results of previous works with their own statistics but the same DSP number to our example FTDL design. As a result, FTDL achieves $7.7\times$ and $7.1\times$ processing throughput to the baseline design; $2.5\times$ and $2.3\times$ to the state-of-art design. The notable performance improvement is contributed from both the high operating frequency of our FPGA layout-aware design and the high hardware efficiency achieved in the compilation. On the power aspect, although the power reaches 45.8 W due to the 650 MHz operating frequency, the *power efficiency* achieves a competitive value of 27.6 GOPS/W, which is $1.9\times$ energy saving compared to the baseline design.

VI. CONCLUSION

In this work, we propose FTDL, an FPGA tailored architecture for deep learning, to address the scalability issue caused by the architecture-layout (FPGA) mismatch in existing designs. The FTDL framework provides a hierarchical and parameterized architecture design that can operate at a frequency of 650 MHz in most FPGAs with different scales. Further, we develop a compilation tool to schedule different DL workloads to the FPGA architecture automatically and achieve high hardware efficiency. Taking the advantages of both high operating frequency and hardware efficiency, FTDL realizes significant improvement in MLPerf benchmarks. FTDL achieves significant improvement in terms of performance and power efficiency on ImageNet with GoogLeNet and ResNet50, compared to the existing designs. With a flexible design, we expect FTDL will be able to serve as the underlying execution platform, which is combined with other algorithm level acceleration techniques such as model compression and quantization, to accelerate DL applications at scale.

ACKNOWLEDGMENT

This work was supported in part by the Croucher Foundation (Croucher Innovation Award 2013), the Research Grants Council of Hong Kong grant number CRF C7047-16G, GRF 17245716, and the US National Science Foundation CRII 2000722.

REFERENCES

[1] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *ISCA*, 2017.
[2] Z. Liu *et al.*, "Throughput-optimized FPGA accelerator for deep convolutional neural networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 3, 2017.
[3] S. I. Venieris and C.-S. Bouganis, "Latency-driven design for FPGA-based convolutional neural networks," in *FPL*, 2017.
[4] L. Lu, Y. Liang, Q. Xiao, and S. Yan, "Evaluating fast algorithms for convolutional neural networks on FPGAs," in *FCCM*, 2017.
[5] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "An automatic RTL compiler for high-throughput fpga implementation of diverse deep convolutional neural networks," in *FPL*, 2017.
[6] R. Shi *et al.*, "E-LSTM: Efficient inference of sparse lstm on embedded heterogeneous system," in *DAC*, 2019.
[7] Y. Ma *et al.*, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 7, 2018.
[8] Y. Guan *et al.*, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *FCCM*, 2017.
[9] X. Wei, *et al.*, "Automated systolic array architecture synthesis for high throughput cnn inference on FPGAs," in *DAC*, 2017.
[10] Y. Ma, M. Kim, Y. Cao, S. Vrudhula, and J.-s. Seo, "End-to-end scalable fpga accelerator for deep residual networks," in *ISCAS*, 2017.
[11] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," *arXiv preprint arXiv:1911.02549*, 2019.
[12] J. Park and W. Sung, "FPGA based implementation of deep neural networks using on-chip memory only," in *ICASSP*, 2016.
[13] E. Nurvitadhi *et al.*, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in *FPGA*, 2017.
[14] J. Fowers *et al.*, "A configurable cloud-scale DNN processor for real-time AI," in *ISCA*, 2018.
[15] *Virtex-7 T and XT FPGAs Data Sheet FPGA Data Sheet: DC and AC Switching Characteristics (DS923)*, Xilinx, Inc., 2019, v1.28.
[16] A. Samajdar, T. Garg, T. Krishna, and N. Kapre, "Scaling the cascades: Interconnect-aware FPGA implementation of machine learning problems," in *FPL*, 2019.
[17] J. Wong *et al.*, "Ultra-low latency continuous block-parallel stream windowing using FPGA on-chip memory," in *FPT*, 2017.
[18] R. Shi *et al.*, "A real-time coprime line scan super-resolution system for ultra-fast microscopy," *IEEE Transactions on Biomedical Circuits and Systems (TBioCAS)*, vol. 13, no. 4, 2019.
[19] A. Canis, J. H. Anderson, and S. D. Brown, "Multi-pumping for resource reduction in FPGA high-level synthesis," in *DATE*, 2013.
[20] K. Chandrasekar *et al.*, "DRAMPower: Open-source DRAM power & energy estimation tool," 2012. [Online]. Available: http://www.drampower.info
[21] Y. Ma *et al.*, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *FPGA*, 2017.