In-Package Domain-Specific ASICs for Intel® Stratix® 10 FPGAs: A Case Study of Accelerating Deep Learning Using TensorTile ASIC

Eriko Nurvitadhi, Jeffrey J. Cook, Asit Mishra, Debbie Marr, Kevin Nealis, Philip Colangelo,
Andrew Ling, Davor Capalija, Utku Aydonat, Aravind Dasu, Sergey Shumarayev
Intel Corporation

***Abstract*— FPGAs or ASICs? FPGAs are extremely flexible while ASICs offer top efficiency. We believe that FPGAs and ASICs are better together, to offer flexibility and efficiency. We propose single-package heterogeneous 2.5D integration of FPGAs and ASICs, using Intel's Embedded Multi-Die Interconnect Bridge (EMIB). Since the ASICs are separate chips from the FPGA, this approach (1) does not change FPGA fabric, allowing re-use of existing ecosystems (FPGA chips, packaging, boards, software), and (2) allows freedom in ASIC design (area/freq/process/etc unconstrained by FPGA fabric). Intel® Stratix® 10 FPGAs already have EMIBs, enabling single-package integration with other chips, or "tiles". We propose leveraging them to mix-and-match any domain-specific ASICs with Stratix10 FPGAs. In particular, this work focuses on deep learning (DL) domain, which demands efficient tensor (matrix/vector) operations. We propose TensorTile ASICs for Stratix10 FPGAs to provide ASIC-level tensor performance, while relying on FPGA's flexibility for application-specific operations (e.g., Winograd). Our evaluation shows: (1) a small TensorTile offer much better tensor throughput than a large Stratix10-2800 FPGA; (2) FPGAs and TensorTiles mix-and-match provide scalable solutions (e.g., ~69 peak INT8 TOPs with 1xTensorTile+small Stratix10-400 FPGA, to ~194 peak FP16 TOPs with 6xTensorTiles+large Stratix10-2800); (3) AlexNet performance (performance/Watt) of Intel's DL FPGA design improved by 4x (3.3x) when enhanced with 2xTensorTiles.**

I. Introduction and Background

The problem: how to improve FPGA efficiency? FPGAs are efficient programmable solutions. FPGA fabric is very customizable, down to bit-level granularities, which allows extracting various levels of parallelisms. As such, FPGAs can be more efficient than other programmable solutions (e.g., better than GPU in deep learning [5]).

However, FPGA fabric's programmability comes at a cost. For an application, an ASIC is an order of magnitude more efficient than FPGA [2]. Nevertheless, an ASIC is inflexible, as it only supports the specific function it is designed for. Making an ASIC more general would require adding more programmability to the ASIC design, leading to inefficiencies. Hence, "general-purpose" programmable ASIC is not always better than FPGA. Ideally, we want to combine the best of FPGAs and custom ASIC accelerators.

Previous solution: "general-purpose" hard blocks in FPGA fabric. Indeed, in an effort to achieve the best of FPGAs and ASICs, existing state-of-the-art FPGAs already integrate custom ASIC blocks (e.g., DSPs for math operations) to improve FPGA efficiency (Figure 1a). However, these ASIC blocks are integrated inside the FPGA fabric. Hence, it necessitates modifications to the FPGA fabric. Since an FPGA fabric architecture requires tremendous effort to develop, it is typically used for the entire FPGA product family of the same generation. So, ASIC blocks integrated within FPGA fabric need to target the general market for FPGAs. Moreover, the scope of the ASIC blocks are limited since they need to comply with FPGA fabric integration constraints (e.g., meet certain delay, size, process technology, etc). Beyond the hardware, the FPGA software tools also needs to be modified to support targeting such modified FPGA fabric.

Proposal: Multi-Chip Integration of FPGAs and ASICs as System-in-Package (SiP). We propose synergistically composing FPGAs and domain-specific ASICs, or "tiles", as multi-chip system-in-package (SiP) solutions (Figure 1b), using technologies such as [3][4][9]. This novel architecture paradigm offers FPGA flexibility and ASIC efficiency. An application domain typically contains operations that are *shared* across the domain and *specific to each application*. Here, the domain-shared operations are implemented on ASIC(s), while the application-specific ones on FPGA(s). The ASICs and FPGAs are integrated using technologies such as Intel's EMIB [3][4], resulting in improved performance and efficiency over "FPGA only" solution, while being more flexible/programmable than "ASIC only" solution.

Furthermore, this approach is extremely cost effective, fast to develop, and scalable due to the following reasons.

(1) Re-use of collaterals (Software, Hardware). We are not changing the FPGA fabric, but instead adding ASICs next to the FPGA chip via efficient links, such as Intel EMIB. So, existing FPGA ecosystems can be reused (e.g., FPGA chips, packaging, boards, software), which save tremendously on costs and development time.

The Stratix 10 FPGA family is already made with EMIB extensibility to provide SiP solutions [4]. It offers

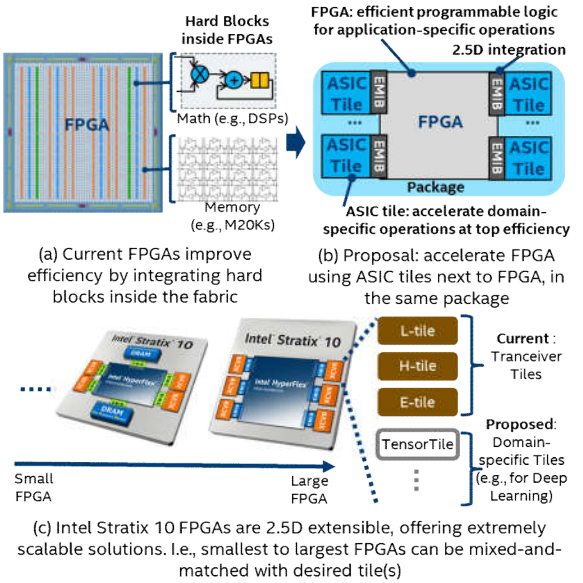

Figure 1. Proposed single-package multi-chip integration of FPGA and domain-specific ASIC accelerator tiles.

myriad of options, from small to large FPGAs, with different numbers of EMIB connections to different "tiles". Each FPGA in the family has at least an EMIB link. These links utilize efficient interfaces (AIB, UIB) [4], and offer data streaming with ~1 Tbps bandwidth.

(2) Scalability/versatility through customized solutions. There is freedom of choice of ASIC tiles to use (e.g., a library of already-available "off-the-shelf" tiles, or new custom tiles). These tiles are not constrained by FPGA in-fabric integration constraints. ASIC area, frequency, and process, etc can be optimized flexibly (e.g., older process for cost savings, or latest process for performance).

Current Stratix 10 FPGAs offer transceiver tiles. Here, we propose domain-specific ASIC accelerator tiles. The ability to mix-and-match any Stratix10 FPGA with set of tiles leads to a versatile and scalable solutions (Figure 1c). One can use a small FPGA with one tile (e.g., embedded), or a large FPGA with 6 tiles (e.g., data centers). Generally, arbitrary FPGAs and ASICs can be composed, resulting in many possible integration options.

Deep Learning Case Study. This paper presents a case study for the Deep Learning (DL) domain. DL is at the forefront of Artificial Intelligence revolution. DL based on Deep Neural Networks (DNN) heavily rely on tensor computations (i.e., matrix/vector multiply/accumulate). To this end, many have proposed accelerators for DL, based on FPGAs [1][5][7][10][14][15], GPUs [13], and ASICs [6]. They offer improved tensor processing. FPGA's fine-grained spatial configurability can be especially appealing, due to the need for extreme customizations in DNN solutions (e.g., as elaborated in [5]). There are industry trends to use FPGAs for DL, from data centers (Microsoft Brainwave [7]) to automotive (Intel GO™ [8]) applications. Nevertheless, FPGA's flexibility comes at a cost. A fixed-function ASIC tensor accelerator may offer top efficiency, but is less flexible than FPGAs. We propose integrating FPGAs with DL accelerator ASICs.

TensorTile: Tensor ASIC Accelerator for Stratix 10. We present a novel ASIC tile, called TensorTile. It complements FPGAs to execute commonly used tensor operations in DNNs with ASIC-level efficiency. Stratix 10 enhanced with TensorTiles can flexibly implement application/use-case specific customized DNN functions on FPGAs (e.g., Winograd), while utilizing TensorTiles to deliver optimal tensor performance. Figure 2 shows an example Stratix 10 FPGA running Intel DL Accelerator [1], offloading tensor operations to a TensorTile.

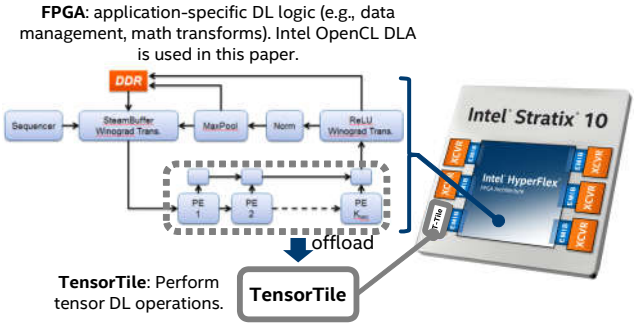

Figure2. TensorTile accelerates FPGA by delivering ASIC efficiency on tensor (e.g., matrix/vector) operations. The figure shows a TensorTile on Intel Stratix10 2800 FPGA accelerating Intel OpenCL Deep Learning solution [1]. FPGA developer uses TensorTile through a well-defined interface, in the same manner as existing hard FPGA blocks. TensorTile can be used with any FPGA design on any tensor applications (e.g., deep learning, wireless, cognitive radio, etc). Different variants/number of tiles can be integrated to offer scalable solutions (e.g., a small FPGA and 1 tile can offer 69 INT8

Our evaluation shows: (1) a small TensorTile (10s in mm^2, 14nm process) offer much better tensor throughput than a large Stratix10-2800 FPGA; (2) FPGAs and TensorTiles mix-and-match provide scalable solutions (e.g., ~69 peak INT8 TOPs with 1xTensorTile+small Stratix10-400 FPGA, to ~194 peak FP16 TOPs with 6xTensorTiles+large Stratix10-2800); (3) AlexNet performance/Watt and performance of Intel's DL OpenCL Stratix10 FPGA [1] improved by 3.3x and 4x when enhanced with 2x TensorTiles. Overall, this approach is effective and scalable.

II. TENSOR-TILE: DEEP LEARNING ASIC ACCELERATOR FOR STRATIX 10 FPGAS

A. *TensorTile Overview*

Given the growing importance of DL, this paper proposes TensorTiles ASIC accelerators to complement FPGAs to offer extremely efficient execution of key tensor operations in DL. Unlike "ASIC-only" accelerators, our design does not include support for all DNN operations, since they are left for FPGA to handle. Unlike "FPGA-only" accelerators, TensorTile ASIC gives major efficiency improvements on tensor operations. It can be used with any FPGA design for any tensor-heavy applications (e.g., DL, wireless, cognitive radio, HPC, etc).

TensorTiles can be mixed and matched with a variety of Stratix10 FPGAs to form scalable customized solutions. Figure 3 shows several example use cases, such as maximizing flexibility and efficiency (Figure 3a), freeing up FPGA space to deploy more applications (Figure 3b), and maximizing efficiency while maintaining sufficient flexibility through the use of smaller FPGA (Figure 3c).

Using TensorTile. FPGA developer use TensorTile through a well-defined interface, in the same way as using existing FPGA components in Quartus library. Developers access TensorTile(s) through Quartus library to be included in their design. Note that any other new tiles (for other domains) can be included in the same fashion.

Operations supported by TensorTile. TensorTile supports key matrix/vector tensor multiply/accumulate operations commonly used in deep learning domain. We support low precision operations, including below 8-bit, such as INT4. Very low precision DNNs have recently been shown to deliver good accuracies (e.g., [11]).

Examples of operations suited for FPGA. On the

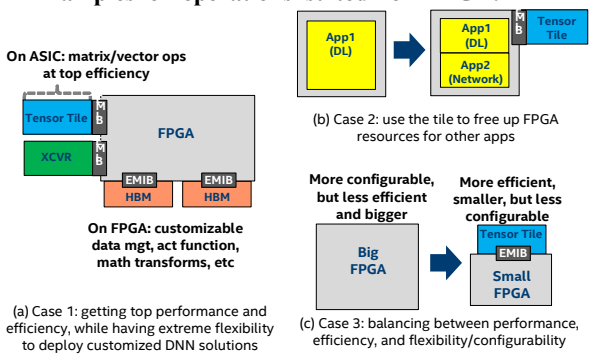

Figure 3. Examples of integrating Stratix 10 FPGA with TensorTiles to offer very versatile solutions across use cases.

other hand, algorithm optimizations such as FFT and Winograd transforms are good only for certain convolution sizes and therefore is application-specific. Similarly, data management from/to off-chip memory can be application specific with different formats and/or data layouts (e.g., images vs. speech data). This type of application-specific operations are better suited for FPGA.

B. *TensorTile Interface and Architecture*

Interface. TensorTile ASIC interfaces to the FPGA chip in a prototypical fashion as existing external hard blocks (e.g., PCIe), except connections are done via chip-to-chip links (i.e., EMIB). The FPGA sends commands and data to tell the ASIC what to do. The ASIC performs the commands and provides results back to FPGA. An FPGA designer interfaces with TensorTile by instantiating a "wrapper" block via Quartus library, which connects to the appropriate FPGA pins associated with the TensorTile. Then, the FPGA designer could implement any desired logic to command the ASIC, provide data to compute, and consume the ASIC results. In particular, TensorTile uses a multiple bi-directional streaming interfaces (s-ifcs), as shown in Figure 4a. It can support a "fully unified mode" where an FPGA application can use the entire TensorTile through all the s-ifcs, or a "clustered mode" where the s-ifcs are divided across multiple FPGA applications. (Each application gets a portion of the TensorTile).

Architecture. TensorTile is based on a systolic array of processing elements (PEs), as shown in Figure 4a. The array is divided into multiple clusters, each connects to a streaming interface (s-ifc). A cluster consists of a systolic chain of PEs to perform computations, and a data management unit (DMU) to transfer data between FPGA chip, PEs, and neighboring cluster(s). As shown in Figure 4b, a PE contains dot product engines and a scratchpad to feed them with PE-local data. It also contains feeder and drainer blocks to receive input and send output. The DMU also contains feeder and drainer blocks, as in Figure 4c. Figure 4d shows a 14nm 4-PE design we placed/routed.

How it works. Input feature map (IFM) is convoluted against filters to produce output feature map (OFM), and computation for each filter is assigned to a PE. First, the filters are loaded into each PE's scratchpad. Then, IFM blocks are streamed to the PEs. As the PEs receive the IFMs, they perform dot product operation against the filter in their scratchpads, and keep track of running sums. As OFM elements are computed, they are drained from the PEs into the FPGA via a chain of drainer blocks.

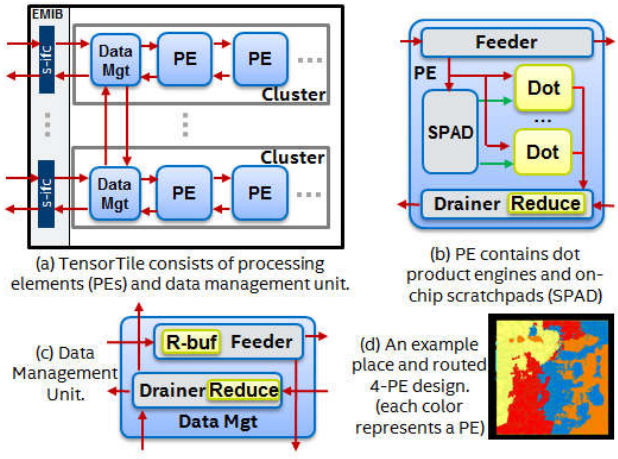

Figure 4. TensorTile Architecture. Overall architecture is shown in (a), PE internal in (b), and Data Management Unit in (c). Then, (d) depicts an example place & routed 4-PE design.

Extracting parallelism. To be able to extract various levels of parallelism and scale to more PEs, we incorporate reduction units in each drainer block. This facilitates reducing partial sums within PEs, across PEs, and across clusters. Consequently, it allows taking advantage of multiple dimensions of parallelism.

Data reuse. To optimize reuse during convolution, the feeder in DMU contains reuse buffer (R-buf in Figure 4c). It keeps a block of IFM at DMU and re-uses it across multiple convolution windows computed by PEs.

C. *Example: Using TensorTile to Accelerate Intel FPGA OpenCL Deep Learning Accelerator (DLA).*

Even though TensorTile can be used with any FPGA design, this paper evaluates TensorTile with Intel OpenCL DL Accelerator (DLA) [1], as depicted in Figure 2. DLA consists of components to move data between the FPGA and off-chip memories, and to perform DNN functions (Pool, Norm, ReLU, Winograd). It also has processing elements (PEs) for tensor dot products. A sequencer unit controls the overall execution of the architecture.

TensorTile(s) naturally extends the DLA architecture, where PEs that are currently on FPGA can be (1) offloaded to TensorTile to free up FPGA resources, or (2) augmented by further tensor processing done by TensorTile.

In practice, other existing DL accelerators also rely on dot product PEs for tensor processing (e.g., [7][10][15]). Hence, TensorTiles can be used in similar fashion with any of these existing FPGA DL accelerators.

TABLE I. EXPLORING THE DESIGN SPACE OF TENSORTILES, FOR SMALL AREA (10S IN MM2) AND POWER (<15W) ON INTEL 14NM. (*CLASS C IS OPTIMIZED FOR COMPUTE AND S FOR ON-CHIP STORAGE)

Data Type	Tile Class*	On-chip Storage (MBs)	Performance (TOP/s)	Efficiency (TOP/s/W)
FP16	S	10	20.7	1.7
	C	7.3	30	2.6
INT8	S	11	46	3.5
	C	8.4	69	5.2
INT4	S	10.7	87	7
	C	8.4	138	10.4

III. EVALUATION

A. *Tensor-Tile performance and efficiency*

First, we explored TensorTiles design space to study the range of peak performances and on-chip storage across variety of tiles. There is large design space of FPGA/ASIC integration because multiple EMIB-links can be connected to arbitrary size chips. Here, we investigated small and lightweight TensorTiles that target 10s in mm^2 area with <15 Watts of power. It is possible to scale up TensorTiles (more area, power). Such study is left for future work.

Peak Performance and On-chip Storage. We studied the TensorTiles in Table 1, with varying precisions (FP16-INT4), compute throughput, and on-chip scratchpads (SPADs). The optimal tile depends on target applications. E.g., data-intensive RNNs may prefer more SPADs. We implemented our designs in RTL and synthesized them to 14nm Intel process. As shown in Table 1, the tiles offer wire range of performance (20-138 TOP/s).

Peak Performance and On-chip Storage of Stratix 10 FPGAs with TensorTiles. TensorTiles can be integrated with any FPGA in Stratix 10 family. Figure 5 shows peak performance and performance/watt for several possible combinations. Included are two FPGAs, large Stratix 10 2800 and medium Stratix 10 2100 with HBMs. We show baseline FPGA with no tile, with 1 tile, and with maximum tiles (6 tiles for 2800 FPGA, 4 tiles for 2100 FPGA). The Stratix 10 FPGA performance baseline is estimated using the method in [10]. We also include GPU numbers, based on public information. Figure 5(g) shows aggregate on-chip storage size in the system. TensorTiles enhanced Stratix 10 solutions cover a wide range of performance/watt, performance, and on-chip storage. For example, a Stratix10 2800 and 6 tiles offer almost 200 FP16 TFLOPs of peak performance, surpassing the NVIDIA Volta™ GPU with TensorCores [12]. With the versatility of our approach, one could choose optimal combinations of FPGAs with number/types of tiles.

For inference, we focus on low-precisions DNNs. As Figure 5 shows, the scaled up 2800 FPGA with 6 tiles can go up to 851 TOP/s INT4 or 483 TOP/s INT8. On the other end, even one tile and the smaller 2100 FPGA can offer 83 TOP/s INT8 or 152 TOP/s INT4. One could also choose to go even smaller for more constrained use case (e.g., automotive, embedded). While not shown in the figure, one could integrate a small Stratix 10 400 (378K LEs) with a single TensorTile-C tile that can deliver ~69 INT8 peak TOPs and ~138 INT4 peak TOPs.

In overall, Performance/Watts are very compelling as well. FPGAs are traditionally known to be power efficient solution. The TensorTiles adds the capability to further improve FPGA efficiency substantially, while offering peak performance. In our study, each tile uses <15Watts, and all tiles under study deliver on average ~4x more tensor throughput than the large Stratix10 2800 FPGA.

Finally, on-chip storage offered by these tiles are non-trivial. For applications that need large storage (e.g., persistent DNNs [7]), a 2800 FPGA with 6 tiles can offer up to 88 MB aggregated on-chip RAMs to accommodate large footprints needed by modern DNNs. As an example, Figure 6 shows footprint of various Resnets (34 to 152 layers) for batch 1 to 8 commonly used in inference. Even the deepest Resnet-152 with 8 batches can fit in the on-chip storage of the 2800 FPGA with 6 TensorTiles.

B. Case study: Intel OpenCL Deep Learning Accelerator (DLA) with TensorTiles

Second, we studied TensorTiles to accelerate Intel OpenCL FPGA DL Accelerator (DLA) [1]. We compared performance and efficiency of FPGA only, FPGA with TensorTiles, and GPU. We target Stratix 10 2100 and 2800 FPGAs. In each, we studied configuration with 2 and 4 TensorTiles. DLA is modified to use TensorTiles (Figure 2) to perform the most demanding task (vector operations). Other operations (e.g., off-chip data handling, Winograd) are done in FPGA, taking advantage of FPGA's flexibility.

We conducted two case studies, based on prior published DLA work [1][10]. Using these "FPGA-only" DLA baselines, we re-target their designs to use TensorTiles. To scale up to multiple TensorTiles, we instantiated multiple DLA instances (minus the PE array), each one offloads tensor operations to its own TensorTile. These studies show the benefits of FPGA flexibility. The first study targets FP16 AlexNet with Local Response Normalization and Winograd [1]. The second study targets INT4 AlexNet with batch norm and no Winograd [10].

We considered TensorTiles from the previous sub-section. For performance modeling, we used DLA framework from [1], which was enhanced to model TensorTile (e.g., FPGA and TensorTile ASIC links). The framework also allows exploring design parameters, which we used to find optimal configuration. We ended up with a configuration based of TensorTile-S (~20TFLOP/s FP16).

Figure 7 shows the results of our case studies. For comparison, we also include GPU results [12][13]. Averaging across two case studies, two TensorTiles offer

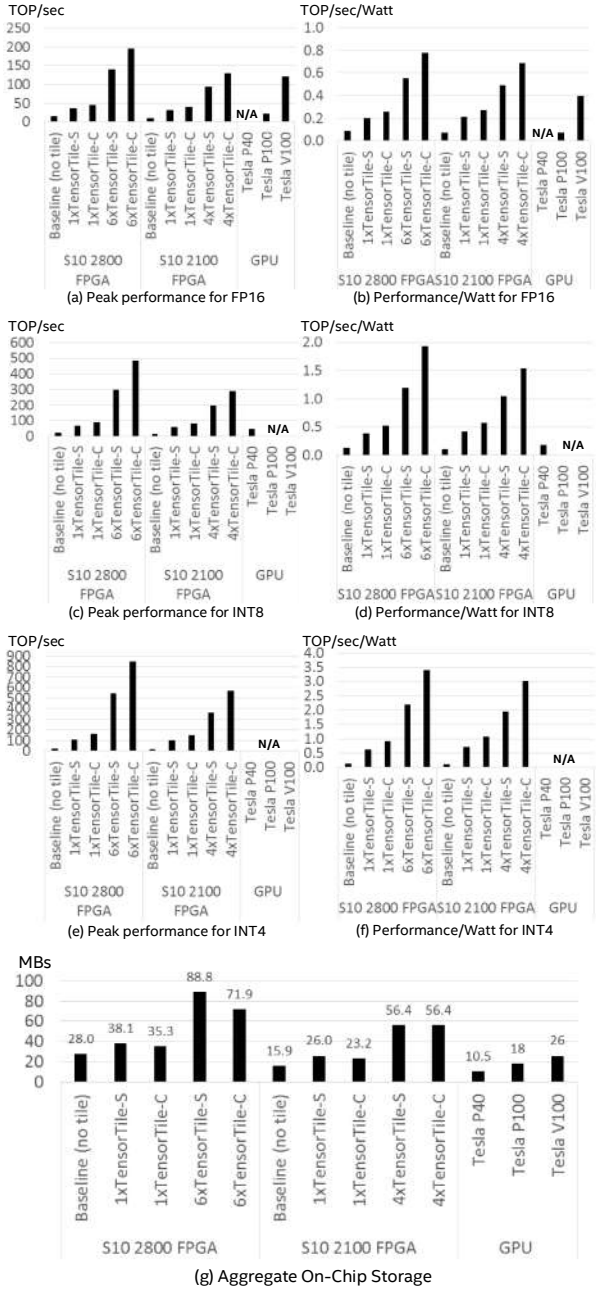

Figure 5. Exploring the design space of Stratix 10 FPGAs with TensorTiles, and comparison with FPGAs and GPUs.

~3.8x and ~3.1x improvements in performance and performance/watt relative to FPGA with no tiles. Four TensorTiles increase these improvements to 7.6x and 5.3x.

Furthermore, going below 8-bit precision seems to be a compelling solution. E.g., Stratix 10 2800 with 4 tiles with INT4 precision can achieve 173K img/sec on AlexNet, substantially higher than the GPUs (Figure 7(a)).

In comparison to GPUs, FPGAs with TensorTiles offer not only better efficiency (performance/watt), but also more versatility and scalability. For the important low-batch inference, even Stratix 10 without TensorTiles are already quite compelling relative to GPUs. The use of TensorTiles offer further performance and efficiency. With just 2 TensorTiles, Stratix 2100 and 2800 based solutions provide better performance and efficiency than Titan X, and comparable to Volta GPUs. Scaling to 4 tiles allows surpassing Volta GPU's performance and efficiency.

In overall, as latency-bound uses cases continues to grow (e.g., [7]), FPGAs and/or FPGAs with TensorTiles are more favorable than GPUs. For throughput oriented use cases (with large batch size), FPGAs with TensorTiles allow scaling up to beyond the peak throughput of GPUs.

IV. Related Work

Existing FPGAs. Existing FPGAs have hard ASIC blocks (e.g., DSPs for math) in their fabric. However, changing FPGA fabric requires significant effort and costs. The functionalities supported also has to be general-purpose enough across FPGA market. The proposed approach does not require any fabric modifications.

Non-FPGA accelerators. Compared to GPUs [12][13], FPGAs are more flexible as it allows finer grained programmability. Our approach improves the existing FPGAs by adding domain-specific ASICs to them.

Compared to existing ASICs (e.g. [6]), our approach leverages existing FPGAs for flexibility/programmability. An "ASIC-only" solution is inflexible. A more general ASIC with programmability can incur overheads in design complexity, efficiencies, and effort/costs.

Deep Learning Accelerators. There are many existing deep learning accelerators on ASIC, FPGA, and GPU (e.g., [1][5][6][7][8][10][12][13][14][15]). To our knowledge, this paper is the first to synergistically combine FPGAs and domain-specific ASICs through multi-chip integration.

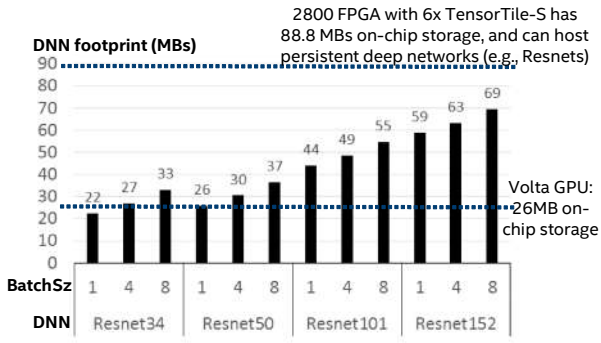

Figure 6. Required on-chip storage sizes to make Resnets persistent, for various depths (34-152) and batch sizes (1-8). The large on-chip storage of 2800 FPGA with 6xTensorTiles can accommodate all of these Resnet configurations. In contrast, even the latest Volta GPU only has 26MB of on-chip storage, and can only fit batch 1 configuration of Resnet 34 and 50.

V. Conclusions

We propose integration of FPGAs and domain-specific ASICs in a single package, using Intel's EMIB. This approach does not require FPGA fabric modifications, and can leverage Intel Stratix 10 FPGAs' already built with EMIBs. Specifically, we proposed TensorTiles ASICs to complement FPGAs for DL applications. Evaluation shows TensorTiles are very efficient and scalable. We reported case studies using TensorTiles to accelerate Intel's OpenCL FPGA DL Accelerator. Lastly, integrating FPGAs and ASICs can also be applied to other domains.

References

[1] U.Aydonat, S. O'Connell, D. Capalija, et. al., "An OpenCL Deep Learning Accelerator on Arria 10," ISFPGA 2017.

[2] I. Kuon, J. Rose, "Measuring the Gap between FPGAs and ASICs," ISFPGA 2006.

[3] Intel EMIB. intel.com/content/www/us/en/foundry/emib.html

[4] S. Shumarayev, "Stratix 10: Intel's 14nm Heterogeneous FPGA System-in-Package (SiP) Platform," HotChips, 2017.

[5] E. Nurvitadhi, et. al., "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?," ISFPGA 2017.

[6] N. P. Jouppi, C. Young, N. Patil, et. al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," ISCA 2017.

[7] E. Chung, J. Fowers, "Accelerating Persistent Neural Networks at Datacenter Scale," HotChips, 2017.

[8] Intel® GO™ Automotive Solutions. www.intel.com

[9] TSMC CoWoS. www.tsmc.com

[10] P. Collangelo et.al., "Exploration of Low Numeric Precision Deep Learning Inference Using Intel FPGAs," FCCM 2018.

[11] A. Mishra, E. Nurvitadhi, J. Cook, D. Marr, "WRPN: Wide Reduced-Precision Networks," ICLR 2018.

[12] O. Giroux, L. Durant, "Inside Volta", GPU Tech. Conf., 2017.

[13] S. Migacz "8-bit Inference with TensorRT", GTC, 2017.

[14] E. Nurvitadhi, et. al., "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," FPL 2016.

[15] D. Moss, et. al, "A Customizable Matrix Multiplication Framework for the Intel HARPv2 Xeon+FPGA Platform - A Deep Learning Case Study," ISPFGA 2018.

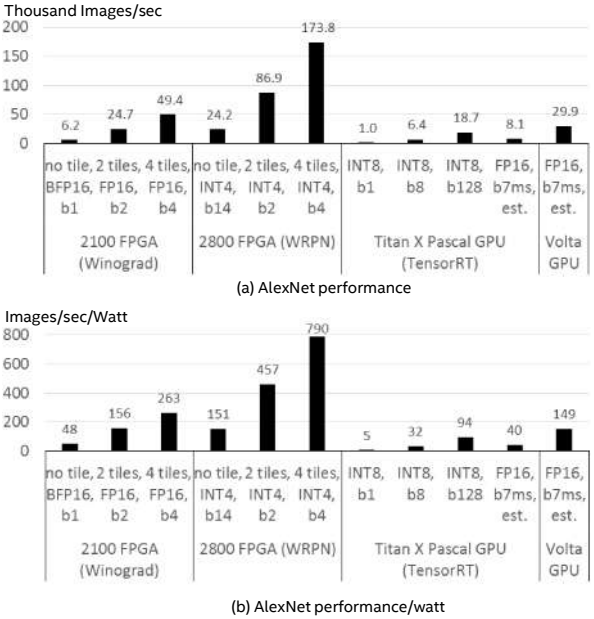

Figure 7. Alexnet Performance and Performance/Watt from case studies using Stratix 10 FPGAs and TensorTiles. We also include published GPU results [12][13]. The FP16 results are estimated based on published Resnet-50 GPU results [12], scaled by the number of ops in Resnet-50/Alexnet. The results are for max batch size within 7ms latency (noted as "b7ms").