Centaur: A Chiplet-based, Hybrid Sparse-Dense Accelerator for Personalized Recommendations

Ranggi Hwang Taehun Kim Youngeun Kwon Minsoo Rhu
School of Electrical Engineering
KAIST
{ranggi.hwang, taehun.kim, yekwon, mrhu}@kaist.ac.kr

***Abstract*—Personalized recommendations are the backbone machine learning (ML) algorithm that powers several important application domains (e.g., ads, e-commerce, etc) serviced from cloud datacenters. Sparse embedding layers are a crucial building block in designing recommendations yet little attention has been paid in properly accelerating this important ML algorithm. This paper first provides a detailed workload characterization on personalized recommendations and identifies two significant performance limiters: memory-intensive embedding layers and compute-intensive multi-layer perceptron (MLP) layers. We then present Centaur, a chiplet-based hybrid sparse-dense accelerator that addresses both the memory throughput challenges of embedding layers and the compute limitations of MLP layers. We implement and demonstrate our proposal on an Intel HARPv2, a package-integrated CPU+FPGA device, which shows a 1.7−17.2× performance speedup and 1.7−19.5× energy-efficiency improvement than conventional approaches.**

***Index Terms*—Accelerator, processor architecture, FPGA, machine learning, neural network, deep learning**

I. Introduction

The complexity of deep neural network (DNN) based machine learning (ML) algorithms is scaling up rapidly. As such, GPUs or ASIC/FPGA-based ML accelerators are widely being adopted for accelerating the computationally *dense* DNN layers. Examples include convolutional neural networks (CNNs), recurrent neural networks (RNNs), and multi-layer perceptrons (MLPs), all of which are amenable for hardware acceleration thanks to their highly regular and deterministic dataflow.

While we were able to make significant strides in accelerating these compute-intensive DNN layers, little attention has been paid in addressing the challenges of memory limited *non*-DNN layers in emerging ML workloads. Consequently, we are witnessing these non-DNN layers, especially those that are memory intensive, gradually becoming a more significant performance bottleneck [23], [35], [38]. In particular, ML algorithms employing *sparse embedding layers* exhibit drastically different characteristics than conventional *dense* DNN layers. Figure 1 illustrates the high-level structure of ML applications employing embedding layers, which are being adopted in a variety of application domains such as ads, social networking service, e-commerce, and others. The backbone ML algorithms that power these applications are *personalized recommendation systems*, the most widely deployed ML workload serviced from the cloud. As we study in this paper, embedding layers account for a significant fraction of the inference time of recommendations. Consequently, several hyperscalers such as Google [18], Facebook [23], [54], [63], Alibaba [60], and Baidu [27] all pinpoint to these embedding layers as causing a severe performance bottleneck in production-level personalized recommendation models.

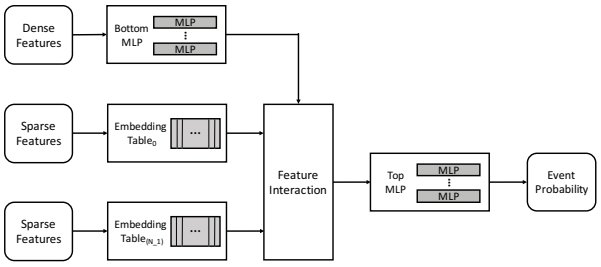

Fig. 1. Topological structure of a DNN-based personalized recommendation model containing sparse embedding layers as the frontend and dense DNN layers as the backend processing step.

In this paper, we focus on addressing the system-level challenges in *deploying* DNN-based recommendation models. Personalized recommendation consists of two key modules: (1) the frontend, *sparse* embedding layers and (2) the backend, *dense* MLP layers. As detailed in Section III, embedding layers consume up to several hundreds of GBs of memory capacity in recommendations, even for inference. Because such requirement is far beyond the physical memory capacity of GPUs (only available with several tens of GBs, 32 GB in NVIDIA V100 [50]), a common practice in deploying these models over the cloud is to utilize the capacity-optimized CPU memory to store the embeddings and utilize the CPU exclusively for inference (`CPU-only`) [23], [25], [54].

Given this landscape, this paper first conducts a workload characterization study on state-of-the-art personalized recommendation models. We identify the following **key challenges** in deploying personalized recommendations on conventional `CPU-only` systems. During the frontend embedding layer stage, multiple embedding vectors are *gathered* from several embedding tables which are subsequently *reduced* over the low-bandwidth CPU memory (Figure 1). Because the aggregate size of the gathered embeddings for inference is much smaller than the size of the embedding tables (e.g., several KBs or MBs of read over several tens of GBs of embedding tables), the embedding gather operations are extremely *sparse* with low spatial/temporal locality, exhibiting high last-level cache (LLC) miss rates (Section III-B). Unlike throughput-optimized

978-1-7281-4661-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ISCA45697.2020.00083

GPUs however, CPUs are primarily optimized for latency with only a handful of concurrent threads and miss status holding registers (MSHRs). As such, we observe that CPUs fail to maximize memory-level parallelism thus significantly under-utilizing memory bandwidth for such sparse embedding gather operations (Section III-C). Consequently, these sparse embedding layers can account for a significant fraction of inference time (up to 79%), causing a performance bottleneck. Another significant challenge with `CPU-only` recommendations is that the compute-intensive MLPs are executed using the low-throughput CPUs, experiencing significant latency overheads. Overall, we identify the *limited memory throughput utilization* of CPU memory systems and the *low computational throughput* of CPUs as the two most significant obstacles in addressing the system-level bottlenecks of personalized recommendation.

To this end, we present `Centaur`, a chiplet-based hybrid sparse-dense FPGA accelerator that holistically addresses the challenges of personalized recommendations. FPGAs have recently had a surge of interest for ML acceleration thanks to their power-efficient and highly programmable nature. However, prior work on FPGA-accelerated ML primarily targets the compute-intensive *dense* DNN layers [4], [12], [20], [42], [44], [45], [48], [53], [58], [59], [64]–[67], so they cannot properly address the challenges of sparse embeddings. Traditionally, the most commonly employed integration tier between the CPU and FPGA is to connect them over the PCIe I/O bus, each with its own *local* physical memory. The FPGA in effect functions as a discrete co-processor device (similar to discrete GPUs) and provides ML acceleration as a service to the CPU via a task *offloading* request. Recent advances in chiplet technology [6], [43], [57] however enabled a more tight CPU+FPGA integration at the *package-level*, providing high-bandwidth and low-latency communication between the CPU and FPGA chiplets over a physically *shared* memory. This allows the FPGA chiplet to directly access the embedding tables stored inside the CPU DIMMs, obviating the need for memory copy operations between host and I/O device memory as required in discrete GPUs or FPGAs. Furthermore, as these package-integration technology matures, we expect an even higher compute density as well as higher inter-chip communication bandwidth [6], [30], [43]. The **key innovation** of `Centaur` is the utilization of this emerging, chiplet-based CPU+FPGA technology to develop a *heterogeneous* accelerator architecture tailored to address the conflicting resource requirements of recommendation models. Concretely, `Centaur` synergistically combines the following two modules for high-performance recommendations:

1) **"Sparse" accelerator for embeddings**: Under our package-integrated CPU+FPGA, the FPGA can directly read (write) from (to) the shared physical memory over the high-bandwidth, low-latency CPU↔FPGA communication links. `Centaur` implements a *sparse* accelerator for high-throughput embedding *gather* and *reduction* operations, directly streaming out the embeddings and reducing them from the CPU memory. This helps improve `Centaur`'s effective throughput in embedding gathers and reductions, achieving superior memory bandwidth utilization for embedding layers.
2) **"Dense" accelerator for GEMMs**: Alongside our sparse accelerator, `Centaur` incorporates a module for accelerating the compute-intensive DNN layers. We design a *dense* accelerator to handle the GEMM (general purpose matrix multiplication) operations such as MLPs or feature interactions, allowing significant latency reduction compared to the baseline `CPU-only` which relies on low-throughput CPU cores for GEMM operation.

```
# N: batch size
# M: average number of lookups per table
# Index array: (i_1, i_2, ..., i_N*M)
# Offset array: (o_1, o_2, ..., o_N)
# Output array: (r_1, r_2, ..., r_N)

00 #pseudo code of SparseLengthsSum operation
01 for a ← 1 to N:
02     r_a ← 0
03     for b ← o_a to o_a+1 – 1:
04         r_a ← r_a + table[i_b]
```

Fig. 2. Functional behavior of `SparseLengthsSum()` in Caffe2 [8], which conducts embedding gathers and reductions.

Overall, our `Centaur` design utilizes the unique properties of package-integrated CPU+FPGAs to demonstrate the merits of a chiplet-based, hybrid sparse-dense accelerator architecture that effectively tackles the performance bottlenecks of personalized recommendation. Specifically, our sparse-optimized accelerator helps overcome the limited memory bandwidth utility of `CPU-only` and achieves significant throughput improvements for sparse embedding layers. Furthermore, `Centaur` improves the performance of MLP layers thanks to the high-throughput FPGA logic. Putting everything together, `Centaur` provides $1.7-17.2\times$ speedup and $1.7-19.5\times$ energy-efficiency improvement than `CPU-only` in deploying end-to-end personalized recommendation models.

II. Background

A. Sparse vs. Dense Layers in Personalized Recommendations

The computer systems community has primarily focused on accelerating the computationally intensive CNNs, RNNs, and MLPs, which exhibit a *dense* and highly *regular* computational property. Because of its highly deterministic dataflow, these dense DNN layers are amenable for hardware acceleration using custom-designed architectures for training and inference [1], [10], [13]–[15], [24], [28], [33], [37], [39]–[41], [51], [52], [55], [56], [59], [61], [62], [64]–[66].

However, emerging ML workloads employing embedding layers exhibit a highly *irregular* and *sparse* dataflow. Figure 2 is a pseudo-code of the `SparseLengthsSum` function implemented in Caffe2 [8], which conducts embedding *lookups* (aka *gathers*) and embedding (vector) *reductions*, widely employed in DNN-based recommendation systems [46]. Millions of vectors called embeddings are stored contiguously inside a table, called embedding (lookup) table, and a sparse index ID

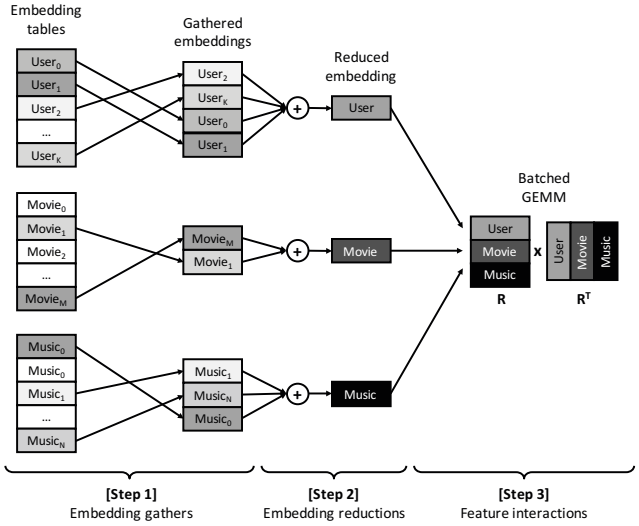

Fig. 3. Illustration of embedding gather and reduction operations, followed by a feature interaction stage. The example assumes three embedding tables are used, each with 4, 2, and 3 gather operations per each table. The feature interaction stage is conducted by a batched GEMM operation, the input of which is collected by concatenating the three reduced embeddings as a tensor.

is used to lookup a unique row from this table. An embedding gather operation takes *multiple* sparse indices as inputs, which do not necessarily point to contiguous rows within the embedding table, to lookup multiple rows from this table. Consequently, an embedding gather operation exhibits a highly sparse and random memory access pattern with low temporal/spatial locality. The embedding vectors gathered from the lookup table can be combined with other vectors using element-wise (addition/multiplication/...) operations, hence performing reductions as illustrated in Figure 3. The reduced embedding vectors go through a feature interaction step to algorithmically capture the complex interaction between different embedding features. While several implementations exists for feature interaction [46], we assume the dot-product based feature interaction method as employed in Facebook's open-sourced deep learning recommendation model (DLRM) [46]. The feature interaction stage in DLRM is implemented by taking the dot-product between all pairs of (reduced) embedding vectors (the batched GEMM operation in Figure 3), the outputs of which are all concatenated with the output vector of the bottom MLP layer (Figure 1). The concatenated vector is then post-processed with the top MLP and fed into a Sigmoid function to calculate an event probability (e.g., the likelihood of a Facebook user clicking an advertisement banner).

B. ML Workloads using Embeddings

An embedding is a projection of a discrete, categorical feature into a vector of continuous real numbers. Under the context of our ML workloads, embeddings are low-dimensional, learned vector representations of feature variables, which have recently shown to be very effective in numerous application domains such as recommendation systems [26], [46], [60], machine translation [19], and automatic speech recognition [5]. A

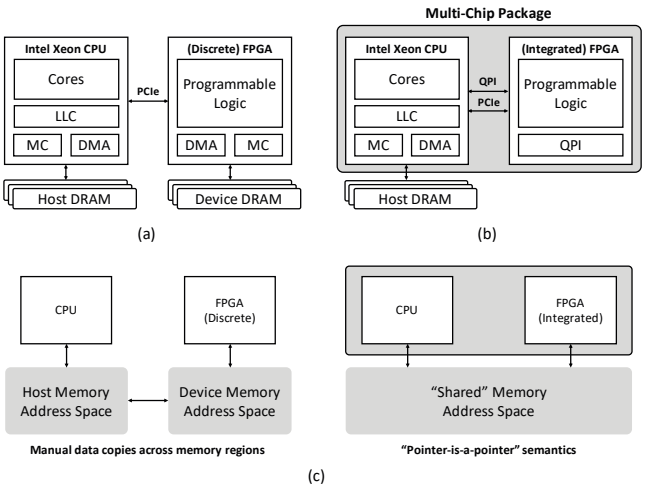

Fig. 4. CPU↔FPGA integration tiers assuming (a) a *discrete* FPGA communicating with the CPU over the PCIe I/O bus, and (b) a *package-integrated* CPU+FPGA housed inside a single CPU socket. (c) The package-level integration of CPU+FPGA enables a *shared* memory address space between the CPU and FPGA, allowing high-bandwidth, low-latency communication between the CPU and FPGA at the hardware level. As this paper utilizes Intel's HARPv2 [29] to demonstrate the merits of chiplet-based CPU+FPGA for recommendations, we assume Intel's technology (e.g., QPI) and nomenclature for the rest of this paper. Nonetheless, the high-level intuitions of our proposal are equally applicable for alternative chiplet-based CPU+FPGA designs.

recommendation system for instance is formulated as a problem of estimating the likelihood of a certain event. A DNN-based recommendation is designed to utilize embeddings to take into account each user and item's learned features and use embedding reductions to interact different features altogether, which is later processed by a backend DNN execution step to extract the probability of a certain event.

C. Discrete vs. Integrated FPGAs for ML Acceleration

While ASICs provide significant energy-efficiency gains than general-purpose CPUs/GPUs for dense DNN layers, they are not able to flexibly cope with the ever-evolving ML algorithm research space. Reconfigurable processor architectures such as FPGAs represent an intermediate design point between the efficiency of ASICs and the programmability of general purpose (CPU/GPU) processors, providing the potential for flexible acceleration of the constantly evolving ML applications [4], [20], [44], [45], [48], [59], [64]–[66]. The most widely employed CPU-FPGA integration strategy is to connect a discrete FPGA card to the CPU over the I/O bus (i.e., PCIe), both of which is equipped with its own local physical memory (Figure 4(a)). Many FPGA boards employ this style of integration because of its extensibility and the high throughput it can provide to the CPU as a co-processor device. A key challenge with such integration tier is that the CPU↔FPGA communication speed is bounded by the narrow PCIe bus bandwidth and its high latency, so the benefits of FPGA acceleration is only provided when its benefits outweigh the task offloading overhead. More recent products therefore employ a more tight CPU+FPGA integration at the *package-level*, allowing the CPU and FPGA chiplets to communicate

TABLE I
RECOMMENDATION MODEL CONFIGURATIONS.

Model	# of Tables	Gathers/table	Table size	MLP size
DLRM(1)	5	20	128 MB	57.4 KB
DLRM(2)	50	20	1.28 GB	57.4 KB
DLRM(3)	5	80	128 MB	57.4 KB
DLRM(4)	50	80	1.28 GB	57.4 KB
DLRM(5)	50	80	3.2 GB	57.4 KB
DLRM(6)	5	2	128 MB	557 KB

at a much higher bandwidth and lower latency than discrete FPGAs (Figure 4(b)), with future designs expected to provide even higher bandwidth and speed using more advanced multi-chip packaging technologies [6], [30], [43]. Another key advantage of integrated CPU+FPGA devices is that they can share a single physical memory, which allows fine-grained FPGA-to-CPU data accesses (and vice versa) at the hardware-level, obviating the latency overheads of traversing through the software stack for data movements (i.e., manual DMA-invoked `memcpy`s across the CPU↔FPGA memory address space, Figure 4(c)) thus reducing overall memory access latency.

III. WORKLOAD CHARACTERIZATION OF DNN-BASED PERSONALIZED RECOMMENDATION SYSTEMS

In this section, we utilize the open-sourced deep learning recommendation model (DLRM) [46] to conduct a detailed workload characterization study on DNN-based personalized recommendations. DLRM comes with several production-level model configurations and we generate six recommendation models that covers the design space of recommendations (as discussed in [23], [46]) by varying the number of embedding tables, number of gather operations per each table, and the total memory usage of embedding tables and MLP layers (Table I). Following prior work [23], [46], each embedding is sized as a 32-dimensional vector as default. A key objective of our characterization study is to root-cause the performance bottlenecks of recommendation models and motivate our hybrid sparse-dense FPGA accelerator design. In the rest of this paper, we assume the `CPU-only` system as our baseline architecture as it is the most commonly deployed system design point for recommendations. We further detail the merits of `CPU-only` for deploying recommendations in Section IV-A.

A. Breakdown of End-to-End Inference Time

Figure 5 shows a breakdown of end-to-end inference latency and normalized execution time when sweeping the input batch size from 1 to 128. There are several interesting observations to be made from this experiment. First, unlike conventional ML applications extensively studied in the computer systems community, *non-DNN* layers such as embedding layers take up significant fraction of execution time on personalized recommendation models. Second, MLP layers still account for a non-trivial portion of runtime, especially when the inference batch size is small. Third, although larger batch sizes increase the latency of both the embedding and MLP layers, MLP layers experience a relatively slower increase in execution time than embedding layers (except for DLRM(6) which is intentionally configured to have a lightweight embedding layer

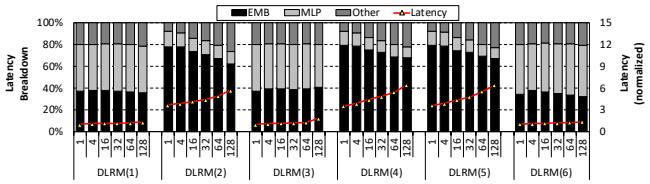

Fig. 5. Breakdown of CPU's inference latency into embedding layers (EMB), MLP layers, and others (left-axis) as a function of batch size, from 1 to 128 (x-axis). The inference latency normalized to the slowest DLRM model with batch size 1 (DLRM(1)) is shown on the right-axis.

followed by a much more compute-intensive MLP layer, see Section V for details of our methodology). This is because large batch sizes tend to help increase the data reuse of MLP weights across the multiple input batches and amortize the cost of uploading weights on-chip (e.g., as detailed in the next subsection, LLC miss rates in MLP layers are never more than 20%), whereas larger batches in embeddings do not translate into better data reuse whatsoever. In other words, large batch sizes simply result in a larger amount of embeddings to be gathered (Figure 2) from the memory subsystem which, depending on the relative execution time of embedding layers with respect to other layers, can result in a proportional increase in execution time. In general, we conclude that DNN-based recommendation systems are severely bottlenecked by embedding layers. Nonetheless, MLP layers also account for a significant portion of execution time, especially when the batch size is small for some configurations.

B. On-chip Caching Efficiency

To better understand the compute and memory bandwidth demands of the aforementioned two bottleneck layers (i.e., sparse embedding layers and MLP layers), we conduct a detailed analysis on the CPU's LLC miss rate and MPKI (misses per thousand instructions) while executing embedding and MLP layers (Figure 6). In general, embedding layer's LLC miss rate shows high sensitivity to input batch size with an increasing number of LLC misses as batch size is increased. The reason behind embedding layer's high LLC miss rate is as follows. A unique property of embedding tables is that its size can be in the order of several tens to hundreds of GBs [23], [38], [54]. This is because the total number of embedding vectors within a table increases proportional to the number of users/items (e.g., total number of users registered or movies serviceable in YouTube/Netflix). As such, the embedding gather operations over such high-capacity embedding tables are extremely *sparse* with little spatial/temporal locality. Now, the aggregate size of the gathered embeddings scales up proportional to the batch size (Figure 2), which directly translates into higher memory traffic – but one with low locality. Larger batch sized embedding layers therefore end up more severely pressurizing the LLC, leading to larger number of LLC misses and higher MPKI (Figure 6).

In terms of the MLP layers, the LLC miss rate of these layers exhibit relatively less sensitivity to the input batch size because the aggregate model size of the MLP layers in all our

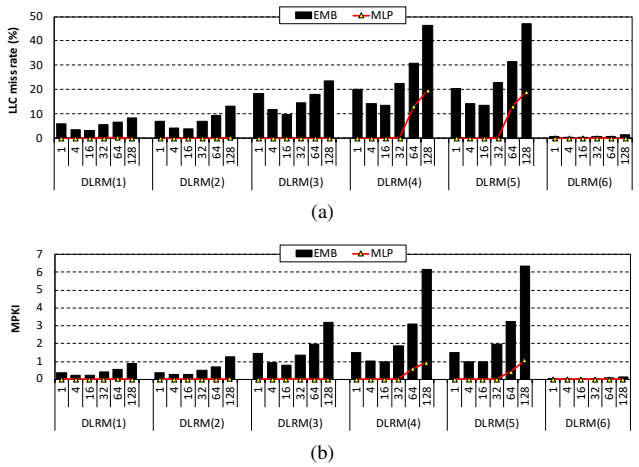

Fig. 6. Effect of executing embedding (EMB) and MLP layers on (a) LLC miss rate and (b) MPKI as a function of batch size (from 1 to 128). We use Callgrind [47] to collect the profiled statistics used for these experiments.

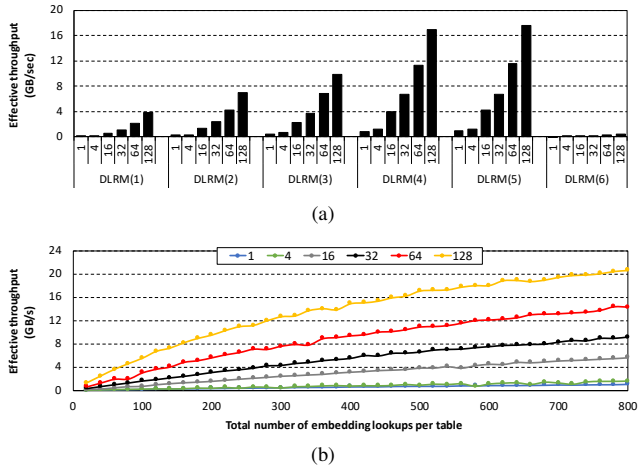

Fig. 7. (a) Embedding layer's effective memory throughput for embedding gathers and reductions as a function of input batch size (from 1 to 128). To quantify its sensitivity to the number of embeddings gathered, the effective throughput of a single table configuration in DLRM(4) is plotted in (b) when sweeping the total number of embeddings gathered. As depicted, the effective memory throughput generally grows monotonically as the batch size increases or when the number of embeddings gathered are increased. However, the effective throughput is far below the maximum memory bandwidth, especially with small batch sizes or under realistic number of gathers per table (i.e., typically under 100 gathers per table [9], [17], [22], [36], [38], [46]).

workloads are sufficiently small enough (typically less than 1MB) to be captured inside the tens of MBs of CPU on-chip caches. Therefore, the MLP layers in recommendation models typically exhibit low LLC miss rates ($<20\%$) and low MPKI, exhibiting a compute-limited behavior.

C. Effective Memory Throughput

While sparse embedding layers exhibit a high LLC miss rate and an accordingly high MPKI (compared to MLP layers), we observe that the *"effective" memory bandwidth utilized in gathering embedding vectors* is extremely low. Figure 7 summarizes the memory throughput of gathering embedding vectors while executing embedding layers. To clearly quantify how efficiently memory bandwidth is being utilized for embedding lookups, we measure the *effective memory throughput for embedding layers* by only considering the useful number of bytes transferred in gathering and reducing embeddings (i.e., size of total embedding vectors gathered / latency incurred in executing the embedding layer)[1]. As depicted in Figure 7, the effective memory throughput for embedding layers is far below the maximum 77 GB/sec of memory bandwidth of our baseline CPU memory system (Section V). Recall that a single embedding vector is only in the order of several hundreds of bytes (i.e., 128 bytes with our default 32-dimensional vector), far below the size of an 8 KB of DRAM row buffer. Additionally, each of these vector loads have limited spatial locality due to their sparse and irregular memory access nature. Unlike throughput-optimized GPUs which execute with several thousands of concurrent threads with a large number of MSHRs (e.g., NVIDIA Volta's L1 cache implements the so-called *streaming cache* which allows unlimited inflight cache misses to maximize data fetch throughput [16]), latency-optimized CPUs utilize only tens of threads with a handful of MSHRs. As the aggregate size of the embedding vectors gathered is only in the order of several KBs (low batch) or MBs (large batch) over several tens of GBs of embedding tables, it makes it challenging for CPU architectures to maximize memory-level parallelism and thus memory bandwidth utilization under the sparse, irregular, and fine-grained vector gather operations[2].

[1]Directly measuring DRAM bandwidth utilization using Intel VTune [32] followed similar trends, albeit with smaller numbers than our defined effective memory throughput as subset of gathered embeddings can hit in the cache.

IV. Centaur: A Hybrid Sparse-Dense Accelerator for Personalized Recommendation

We present `Centaur`, a chiplet-based hybrid sparse-dense accelerator that holistically addresses the dual challenges of memory limited embeddings and compute limited MLPs of personalized recommendations. To the best of our knowledge, `Centaur` is the first end-to-end accelerator that tackles both the memory and compute bottlenecks of personalized recommendation models. We first present our motivation for a package-integrated CPU+FPGA platform (rather than ASICs), followed by a description of our proposed architecture.

A. Motivation: Why Package-integrated CPU+FPGAs?

GPUs are currently the dominating processor architecture for ML training because their throughput-optimized design suits well for the (throughput-heavy) algorithmic nature of training. For cloud deployment of recommendation services however, latency-optimized CPUs are the preferred architecture of choice. First, the abundance of readily available

[2]It is possible to achieve more than 50 GB/sec of effective throughput ($>70\%$ of max) in embedding layers when the batch size is larger than 2048 or when the embedding vector dimension is sufficiently large (i.e., more than 1024-dimensional vector). However, such large batch size and wide embedding dimensions is an unrealistic one to assume for inference.

CPUs in today's datacenters makes it an appealing computing platform from a total cost of ownership (TCO) perspective, especially when considering the off-peak portions of the diurnal cycle where CPUs would otherwise remain idle [25]. Second, user-facing inference services (e.g., news feed, advertisement, e-commerce) for recommendations have firm SLA (service level agreement) goals to meet which renders latency-optimized CPUs more suitable than throughput-oriented GPUs. Lastly, recall that sparse embedding layers are significantly memory capacity hungry because the embedding tables can require up to several hundreds of GBs of memory usage (Section III-C). As a result, the bandwidth-optimized 3D stacked memory employed in GPUs or ML accelerators such as Google TPUs [21], [34] cannot store the embedding tables locally inside their physical memory, preventing them from being used for deploying recommendations. Therefore, the vast majority of cloud ML inference services for personalized recommendation are primarily powered using `CPU-only` systems as noted by several hyperscalers [23], [25], [54].

Given this landscape, we observe that package-integrated CPU+FPGAs become a promising solution as it holistically addresses *all* the aforementioned challenges, as detailed below:

1) Package-integrated CPU+FPGAs are minimally intrusive to existing server chassis designs (and therefore the server rack and the overall datacenter) as they are *socket* compatible to existing system nodes. Furthermore, CPUs can still function as a "*host*" from the OS's perspective (unlike GPUs/TPUs which are slave devices). As such, they can be utilized for *non*-ML usages thus enhancing the resource utility for optimizing TCO.
2) The reconfigurable FPGA logic can be utilized to address performance bottlenecks, further reducing inference latency to help satisfy SLA goals and improve QoS.
3) More importantly, the CPU and FPGA both share a single physical memory (i.e., the memory DIMMs within/across CPU sockets) which is based on capacity-optimized DDRx. This allows the FPGA-side accelerator to keep the entire embedding tables in CPU memory as-is and access them directly using the high-bandwidth, low-latency CPU↔FPGA communication channels, a requirement *discrete* GPUs or FPGAs cannot fulfill.

Based on these **key observations**, our `Centaur` architecture utilizes the FPGA's programmable logic area to implement a *heterogeneous* computing device, synergistically combining a sparse accelerator for embedding gathers/reductions and a dense accelerator for GEMM computations. Before we detail the microarchitecture of our sparse-dense accelerator, the next subsection first discusses our proposed chiplet-based CPU+FPGA architecture. We then discuss our proof-of-concept CPU+FPGA substrate, Intel HARPv2 [29], which we utilize to demonstrate our proposal.

B. Proposed Chiplet-based CPU+FPGA Architecture

Figure 8 illustrates our proposed chiplet-based CPU+FPGA architecture, which is designed to be minimally intrusive to existing TCO-optimized server chassis/rack as it is socket-compatible to current systems. The FPGA chiplet has two communication paths to the CPU memory subsystem. The *cache coherent path* utilizes the CPU↔FPGA cache coherent links (denoted as CC-link$_n$) to traverse through the CPU on-chip cache hierarchy first and then to the off-chip memory (via the blue-colored arrows), which can be effective for memory accesses with high data locality. An alternative *cache bypassing path* (via the red-colored arrows), which is more appropriate for our memory-intensive embedding layers, utilizes a separate memory channel interface that completely *bypasses* the CPU caches and directly routes FPGA-side memory requests to the off-chip memory interface. By provisioning the cache bypassing route's communication throughput to be commensurate to (or higher than) the maximum off-chip memory bandwidth, the sparse accelerator of `Centaur` can significantly boost the throughput of embedding layers by conducting vector gathers over this communication channel.

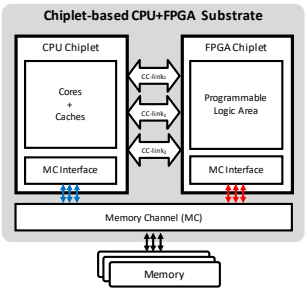

Fig. 8. Proposed package-integrated CPU+FPGA architecture.

Unfortunately, chiplet-based commercialized CPU+FPGA designs are still at an early stage with limited accessibility and functionality. We therefore utilize Intel's HARPv2 [29] as a proof-of-concept substrate to demonstrate the merits of our proposal. As we detail in the next subsection, HARPv2 comes with the cache coherent path (but no cache bypassing route) for CPU memory accesses, so the throughput benefits of our sparse accelerator is constrained by the memory-level parallelism that can be reaped out over the CPU↔FPGA cache coherent path, and accordingly the CPU cache hierarchy. Nonetheless, we use it to conservatively estimate the throughput benefits chiplet-based CPU+FPGAs can provide for recommendations. In the following subsections, we first present the details of our sparse-dense accelerator microarchitecture, followed by a description of its software interface to the overall system.

C. Sparse Accelerator

The key design objective of our sparse accelerator is to enable high-throughput, low-latency embedding gather and reduction operations. Recall that package-integrated CPU+FPGA devices enable the custom-designed FPGA logic to *directly* access the shared physical memory system in fine-grained (64-Byte) cache line granularity via cache-coherent high-bandwidth communication links. Under the Intel HARPv2 platform we assume in this work, a theoretical

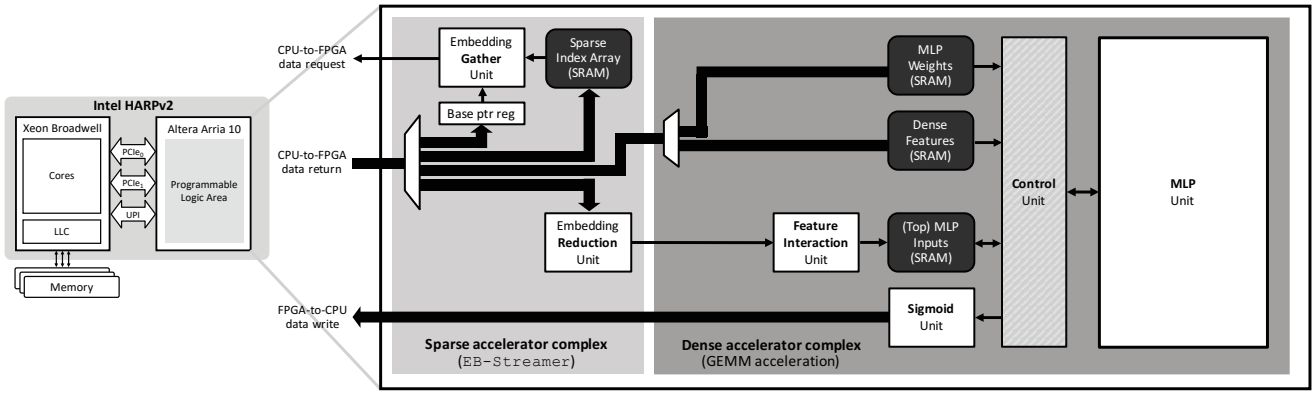

Fig. 9. High-level overview of our proposed `Centaur` architecture. As a proof-of-concept prototype, we utilize Intel HARPv2 to design our hybrid sparse-dense accelerator. The reconfigurable FPGA logic are used to synthesize both the sparse (`EB-Streamer` used for high-throughput, low-latency embedding gathers and reductions) and dense (for high-throughput GEMM computation) accelerators.

maximum uni-directional communication bandwidth of 28.8 GB/sec is provided between the CPU and FPGA using two PCIe links and one cache coherent UPI link. Our sparse accelerator utilize such communication technology to implement an *embedding streaming unit* (henceforth referred to as `EB-Streamer`) that spawns off multiple embedding vector gather operations followed by an on-the-fly reduction operation, in a high-throughput manner. Figure 10 details the microarchitecture of `EB-Streamer`, which contains a base pointer register set (`BPregs`), sparse index SRAM array ($\texttt{SRAM}_{sparseID}$), embedding gather unit (`EB-GU`), and the embedding reduction unit (`EB-RU`). The embedding gathers and reductions are conducted as follows:

1) When system is booted up, the CPU utilizes the MMIO interface to inform the FPGA the CPU memory addresses that point to a) the sparse index array (i.e., the row IDs to gather from the embedding table), b) the embedding table, c) the MLP weights, and d) dense features (to be used as inputs for the bottom MLP). These base pointer values are copied into the `BPregs` to be utilized by the sparse-dense accelerators for both embedding gathers and GEMM operations.
2) Once `BPregs` is initialized, the `EB-GU` utilizes `BPregs`'s base pointer address of the sparse index array to perform a CPU→FPGA read operation which populates the $\texttt{SRAM}_{sparseID}$ with sparse index IDs subject for gather operations. Notice that `EB-GU` is nothing more than an address generator (i.e., base + offset, see Figure 2) which is dominated by logic gates, thus having low implementation overhead.
3) Using the embedding table base address value stored in `BPregs` and the sparse index IDs stored in $\texttt{SRAM}_{sparseID}$, the `EB-GU` starts generating CPU→FPGA embedding gather operations. To maximally utilize CPU↔FPGA communication bandwidth, the `EB-GU` monitors the communication bandwidth utility and aggressively instantiates embedding vector read operations over the PCIe/UPI links, whenever the CPU↔FPGA communication links become available.
4) When the embedding vectors arrive at the sparse accelerator, they are immediately routed to our `EB-RU`. As vector reductions are *in-place* operations, `EB-RU` conducts embedding "reduction" operations on-the-fly whenever the embedding vectors are streamed into `EB-RU`.
5) Once all embeddings are gathered and reduced, the `EB-RU` forwards the reduced embedding vector to the dense accelerator complex.

As embeddings are typically sized as 32-wide vectors, a single embedding vector gather operation is equivalent to a 32×4=128-Byte load instruction. Note that the memory addresses of the multiple embedding vectors subject for gathering are scattered across the memory address space. Consequently, a brute-force, software level data transfer over such fine-grained, irregular data access stream can incur severe latency overheads as each `cpuToFpgaMemcpy()` API execution for a 128-Byte CPU→FPGA read operation must traverse through various layers in the software stack. One of the key advantage of initiating embedding gather operations over the package-integrated CPU↔FPGA channels is that the process of data fetch and retrieval is entirely orchestrated at the hardware level, significantly reducing memory access latency. Furthermore, embedding gathers are conducted while being less interfered and bottlenecked by the CPU's cache hierarchy. As discussed in Section III, embedding gather operations are inherently sparse with extremely low locality, rendering conventional CPU caching mechanism ineffective. Nonetheless, the baseline `CPU-only` system must always traverse through the multi-level on-chip caches for all embedding vector load operations, only to discover that the embeddings to be gathered are (most likely) located in CPU memory. Because the entire embedding gathering process is orchestrated using a handful of threads, `CPU-only` embedding gathers are limited in terms of both parallelism and locality, achieving low memory bandwidth utility (Figure 7). Because our sparse accelerator directly fetches the embeddings over the CPU↔FPGA communication

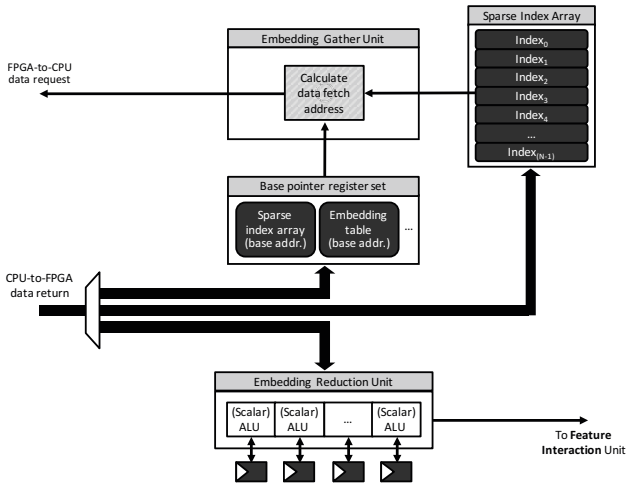

Fig. 10. Microarchitecture of `Centaur` sparse accelerator.

links, `Centaur` can achieve significantly higher memory bandwidth utilization (Section VI-B) and fundamentally address the memory bandwidth challenges of embedding layers.

D. Dense Accelerator

We now present our dense accelerator design, the microarchitecture of which is shown in Figure 11. The primary design objective of our dense accelerator is to speed up the execution of GEMM, the key algorithm that powers both the MLP layers and the batched GEMM operation for feature interactions. We use Altera's FPGA floating-point IP core [3] optimized for matrix multiplications between two square matrices (the FP_MATRIX_MULT module) as key building blocks to construct our dense accelerator complex. A processing engine (PE) in Figure 11 is based on a single instance of the FP_MATRIX_MULT module (configured to handle matrix multiplication between two $[32 \times 32]$ matrices), which we utilize to compose a 4×4 spatial PE array for the MLP unit and another four instances of PEs for the feature interactions. Putting all these together, `Centaur` provides an aggregate computational throughput of 313 GFLOPS operating over 200 MHz. The MLP control unit employs an output-stationary dataflow [11] which tiles the input and weight matrices in $[32 \times 32]$ sizes (to be compatible with the PE's GEMM compute granularity) and broadcasts these tiles across the spatial PE array. The MLP unit then conducts an outer-product among the input and weight tiles using the PE array, which generates the partial sums to be temporally accumulated into the SRAM buffers allocated per each PE (Figure 12). In addition to the GEMM computation units, the dense accelerator complex contains several SRAM buffers to store 1) the MLP weights ($\texttt{SRAM}_{MLPmodel}$), 2) the dense features to be used as inputs to the bottom MLP layers ($\texttt{SRAM}_{DenseFeature}$), and 3) the (top) MLP inputs ($\texttt{SRAM}_{MLPinput}$). The model parameters that are used to execute both top and bottom MLP layers are copied over the CPU↔FPGA communication link using the `BPregs` at boot-time. The MLP weight values

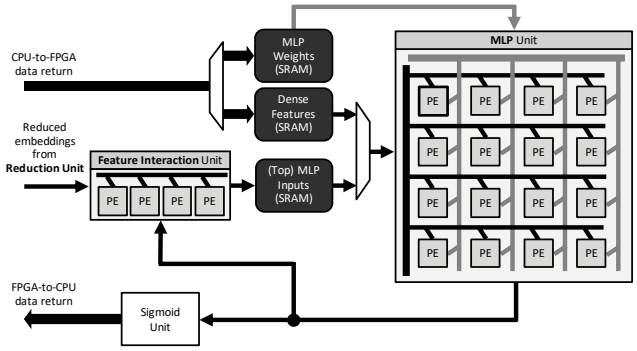

Fig. 11. Microarchitecture of `Centaur` dense accelerator.

remain persistent throughout the entire deployment process, so the overhead of uploading model weights to the FPGA's $\texttt{SRAM}_{MLPmodel}$ is negligible as it is amortized over all future inference requests serviced by `Centaur`. Using these modules, the dense accelerator complex goes through the following steps to finalize the recommendation process.

1) The `BPregs` in the sparse accelerator complex is used to upload the MLP weights into $\texttt{SRAM}_{MLPmodel}$ and the inputs to the bottom MLP layer into $\texttt{SRAM}_{DenseFeature}$. As noted above, initializing the $\texttt{SRAM}_{MLPmodel}$ with model parameters only has to be done once as they remain persistent, whereas $\texttt{SRAM}_{DenseFeature}$ needs to be updated whenever there is a new inference request.
2) The MLP unit first uses $\texttt{SRAM}_{MLPmodel}$ and $\texttt{SRAM}_{DenseFeature}$ to execute the bottom MLP layer, the result of which is forwarded to the feature interaction unit.
3) Once the sparse accelerator forwards the reduced embeddings to the feature interaction unit, the output vector of the bottom MLP layer is concatenated with the reduced embeddings to form a tensor. The feature interaction unit utilizes the concatenated tensor to initiate a batched GEMM computation for feature interactions (Figure 3), the result of which is stored into $\texttt{SRAM}_{MLPinput}$.
4) The outputs of the feature interaction unit, which is read out of $\texttt{SRAM}_{MLPinput}$, is subsequently routed to the MLP unit to execute the top MLP layers using the model parameters stored inside $\texttt{SRAM}_{MLPmodel}$.
5) Once the top MLP layers complete execution, the final results are forwarded to the Sigmoid unit to calculate the event probability. The final result is then copied back to the CPU memory for post-processing.

As the entire dense GEMM computation is orchestrated seamlessly with the sparse accelerator, `Centaur` provides significantly higher throughput and reduced latency in executing dense DNN layers compared to `CPU-only` systems. In the following subsection, we detail the software interface that enables CPU+FPGA integration into the overall system.

E. Software Interface

As the package-integrated HARPv2 platform provides a unified virtual memory address space between the CPU and

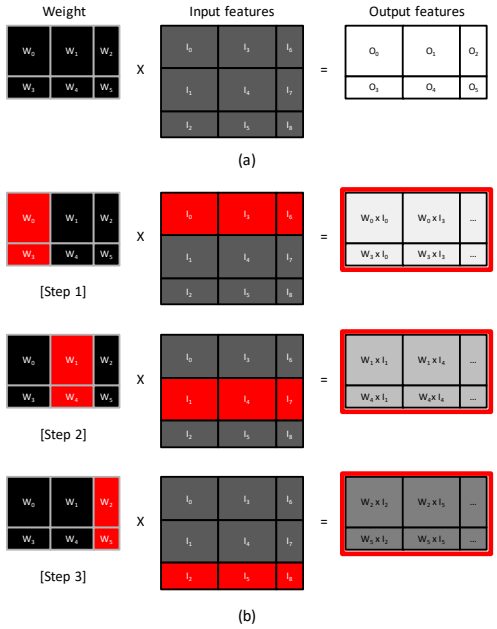

Fig. 12. The output-stationary dataflow in Centaur's MLP unit's (a) GEMM operation. (b) An outer-product between the weight-input tiles generates the output tiles to be accumulated into the intra-PE SRAM buffers. Each PE conducts a $W_m \times I_n$ matrix multiplication operation between the weight and input tiles. In each computation step, a given W_m tile (I_n tile) is broadcasted to all the PEs within its corresponding row (column) using the bus interconnection network within the MLP unit (Figure 11).

FPGA, the CPU+FPGA functions as a single processor as far as the operating system and its applications are concerned, supporting the "pointer-is-a-pointer" like semantics. Concretely, the pointers to the sparse index array, the embedding tables, the dense feature inputs, and others are forwarded to the FPGA using the MMIO interface. As these base address pointers are virtual addresses, the FPGA-side IOMMU (and TLB) translates them into physical addresses when the embedding gather operations are conducted, allowing the FPGA to directly access the CPU physical memory at the hardware level. Compared to invoking multiple software invoked DMA copy operations, such fine-grained hardware level data movement helps reduce average memory access latency, allowing Centaur to achieve superior memory throughput for embedding gathers. Once the base pointer address values for key data structures (e.g., sparse index array, embedding tables, ...) are copied over to the Centaur's BPregs over MMIO, the inference process is entirely orchestrated under the hood at the hardware level. As a result, high-level ML framework (e.g., TensorFlow, PyTorch) can readily employ our proposed architectural solution with minimal changes.

V. Methodology

Evaluation platform. We demonstrate and benchmark Centaur on Intel HARPv2 system containing a Broadwell Xeon E5-2680v4 and Altera Arria 10 GX1150 [29]. At the time of this writing, Intel's HARPv2 platform (released in 2016) is the *only* publicly accessible package-integrated x86

TABLE II
Centaur FPGA resource utilization.

	ALM	Blk. Mem	RAM Blk.	DSP	PLL
GX1150 (Max)	427,200	55.5 M	2,713	1,518	176
Centaur	127,719	23.7 M	2,238	784	48
Utilization [%]	29.9	42.6	82.5	51.6	27.3

CPU+FPGA so we evaluate Centaur using this computing architecture as a proof-of-concept prototype. The entire sparse-dense accelerator is written in SystemVerilog RTL and we use Quartus Prime Pro 16.0 to synthesize, place, and route our design (Table II). We explore three design points of recommender systems. The baseline CPU-only uses HARPv2's Broadwell CPU without the FPGA activated for a fair comparison with Centaur. Aside from Centaur, we also established an additional design point to better cover the design space of recommendation inference systems. While CPUs are the preferred system design point in deploying recommendations (as discussed in Section IV-A), we nonetheless evaluate the performance of a GPU-based system for the completeness of our study. Here, we assume the entire embedding tables are stored in CPU memory so once all the embedding vectors are gathered and reduced by the CPU (using SparseLengthsSum(), Figure 2), the CPU copies them over PCIe to the GPU for GPU-side MLP computation (referred to as CPU-GPU [38]). We utilize NVIDIA DGX-1 [49] for CPU-GPU performance measurements. When estimating CPU's power consumption, we used pcm-power for both CPU socket-level power estimation as well as the power consumed by its memory DIMMs. For GPU power consumption, NVIDIA's nvprof profiling tool has been utilized. For Centaur's CPU+FPGA power measurements, we use pcm-power to measure both the socket-level CPU+FPGA as well as the power consumed by the memory DIMMs. When evaluating energy-efficiency, we multiply the power estimation values with each design-point's end-to-end inference execution time. All performance numbers are measured end-to-end in wall clock time, which is collected after sufficiently warming up the CPU's cache hierarchy.

Benchmarks. We use the open-sourced deep learning recommendation model (DLRM) as our primary benchmark suite [46]. DLRM is configured using the latest PyTorch backend library (version 1.5 nightly build, accessed March 25, 2020) which extracts parallelism using OpenMP and AVX instructions for embedding and MLP layers. DLRM provides three reference model architectures which are used across two different services and have different configurations depending on their use-case. The configurations vary in terms of the number of embedding tables, the number of gathers per each embedding table, total memory requirement of embedding tables, and the number of MLP layers and its dimension size. While maintaining the distinctive characteristics of the default three models, we add three more configurations to better highlight the different compute and memory access behavior of recommendation models, as detailed in Section III. Table I summarizes the six benchmarks we study in this paper. Note

TABLE III
SPARSE VS. DENSE FPGA RESOURCE USAGE.

	Module	LC comb.	LC reg.	Blk. Mem	DSP
Sparse	Base ptr reg.	98	211	0	0
	Gather unit	295	216	0	0
	Reduction unit	108	8,260	0	96
	SRAM arrays	350	98	12.2M	0
	Total	**851**	**8.8K**	**12.3M**	**96**
Dense	MLP unit	40K	131K	2.3M	512
	Feat. int. unit	10K	33K	593K	128
	SRAM arrays	1K	11K	1.6M	48
	Weights	13	77	5.2M	0
	Total	**52K**	**175K**	**9.8M**	**688**
Others	Misc.	587	6K	608K	0

that DLRM(6)'s embedding layer has been artificially scaled down to have a short embedding layer stage with a relatively longer MLP computation step, which we utilize to evaluate Centaur's sensitivity to MLP intensive recommendations.

VI. EVALUATION

This section explores three design points of recommender systems: 1) baseline CPU-only, 2) CPU-GPU, and 3) Centaur. We first discuss the FPGA resource utility of our hybrid sparse-dense accelerator. We then compare the memory throughput and overall performance of CPU-only vs. Centaur, followed by a comprehensive comparison study between all three design points in terms of energy-efficiency.

A. Centaur FPGA Resource Utilization

Table III summarizes how Centaur's sparse-dense accelerator utilizes the various FPGA resources. As the major role of our sparse-optimized accelerator is to perform high-throughput embedding gathers/reductions, the EB-Streamer is designed to incorporate a local sparse index array to be able to seamlessly invoke multiple embedding gather operations in parallel. That is, we employ a large SRAM array to hold many sparse index IDs such that the embedding gather unit can aggressively launch multiple gather operations concurrently, boosting memory-level parallelism and overall memory bandwidth utilization. This is reflected by the sparse accelerator complex using 54% of the block memory bits to store sparse indices, with little usage of the ALMs and DSPs (6% and 12% usage, respectively) as the primary computation conducted inside the sparse accelerator is the address generation for gathers and reductions, both of which can be designed in a lightweight fashion. The dense accelerator complex on the other hand is designed for high computational throughput, so it consumes 88% of the DSPs and 94% of the ALMs, achieving much higher computation throughput than CPU-only systems. As we further discuss in the remainder of this section, such rather skewed, heterogeneous usage of FPGA resources helps Centaur strike a balance that effectively tackles the bottlenecks of memory intensive embedding gathers and compute limited GEMM operations.

B. Effective Memory Throughput for Embedding Layers

CPU-only cannot effectively execute embedding layers because of its low memory throughput in gathering embeddings, spending significant faction of time on this bottleneck

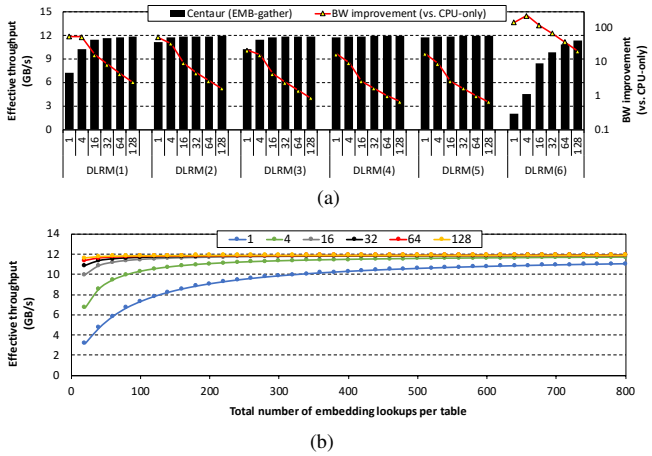

Fig. 13. (a) Centaur's effective memory bandwidth utilized for embedding gathers (left-axis) and its improvements compared to CPU-only (right-axis) as a function of input batch size (from 1 to 128). (b) Centaur's effective memory bandwidth as a function of total number of embeddings gathered from the embedding tables, exhibiting a much rapid improvement in effective throughput than the baseline CPU-only (Figure 7(b)).

layer. Our EB-Streamer significantly improves the effective throughput in gathering embedding vectors, especially for low batches, achieving up to 11.9 GB/sec of throughput (Figure 13). As the maximum possible *effective* uni-directional CPU↔FPGA communication bandwidth is around 17−18 GB/sec in HARPv2, our EB-Streamer achieves 68% of the possible communication bandwidth. Given the highly irregular, sparse data access patterns of embedding gathers, EB-Streamer's high communication bandwidth utility demonstrates the robustness of our embedding gather unit. Because large batches help CPU-only better utilize memory bandwidth (Figure 7(a)), the gap between CPU-only and Centaur's memory throughput gradually shrinks as batch size is increased. In particular, EB-Streamer falls short than CPU-only by 33% for DLRM(4) and DLRM(5) with a large batch size of 128, as EB-Streamer's throughput is constrained by the CPU↔FPGA link bandwidth. As detailed in Section VI-C, such performance overhead for large batches is offset by the high-throughput Centaur's dense accelerator delivers. Note that the effective throughput of EB-Streamer is expected to naturally scale up as CPU↔FPGA communication link bandwidth is increased with the latest high-bandwidth package-level signaling technologies [6], [31], [43], [57]. Overall, Centaur provides an average 27× throughput improvement than CPU-only across our studied configurations, even with our conservatively chosen HARPv2 platform, thus effectively tackling the memory bandwidth limitations of embedding layers. We now discuss the end-to-end performance improvement our Centaur delivers using our sparse-dense hybrid accelerator architecture.

C. Performance

Centaur significantly improves the performance of memory limited embedding layers, thanks to EB-Streamer's high-throughput gather operations. At the same time, the

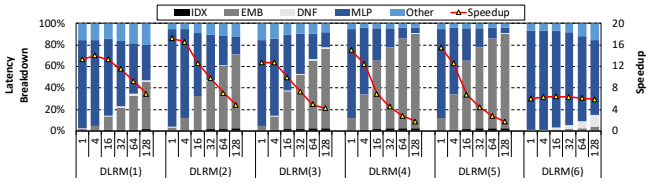

Fig. 14. Breakdown of Centaur's inference time into CPU→FPGA sparse index fetch time (IDX), embedding gathers/reductions (EMB), CPU→FPGA dense feature fetch time (DNF), MLP execution, and others (left axis). The right-axis summarizes the performance improvement Centaur achieves compared to CPU-only.

abundant computation units in dense accelerator complex reduces the latency to execute GEMMs in recommendation models. This allows Centaur to substantially reduce end-to-end latency as it holistically addresses the two most significant bottlenecks of recommendation. Figure 14 shows a latency breakdown of our studied workloads and the resulting performance improvement against baseline CPU-only, achieving $1.7-17.2\times$ end-to-end speedup. Among the six DLRM models we study, five of them are bottlenecked by embedding layers especially under low batches, so the throughput-optimized EB-Streamer helps resolve the system bottlenecks, achieving superior performance improvements. DLRM(6) achieves a modest $6.2\times$ average speedup, which is expected because this model is intentionally configured to have a heavyweight MLP layer with a lightweight embedding layer (Table I). Consequently, the overall performance is relatively insensitive to the improved memory throughput EB-Streamer brings about. Nonetheless, Centaur's dense accelerator still provides significant latency reduction when executing DLRM(6)'s GEMM, achieving substantial end-to-end performance improvement.

D. Power and Energy-Efficiency

So far we have demonstrated the superior memory bandwidth utility and performance of Centaur against the baseline CPU-only. This section provides a comparison study of Centaur against CPU-only and CPU-GPU in terms of power and energy-efficiency. Table IV summarizes the power consumption of our evaluated systems, the methodology of which is summarized in Section V. Compared to the baseline CPU-only or the power-hungry CPU-GPU, Centaur consumes much less power, as the CPU cores mostly remain idle while the FPGA-side sparse-dense accelerator orchestrates the embedding gathers/reductions and the backend MLP computation step in a power-efficient manner. As Centaur achieves superior power-efficiency while also significantly improving end-to-end inference time, the overall energy-efficiency is also significantly improved. Figure 15 provides a summary of the performance and energy-efficiency improvements Centaur brings about. In general, the baseline CPU-only performs better than CPU-GPU on average, achieving $1.1\times$ and $1.9\times$ performance and energy-efficiency improvements. As the CPU-GPU design needs to store the embedding tables inside the CPU memory, the CPU-invoked embedding gathers/reductions must always copy the reduced embeddings to the GPU for MLP acceleration. This causes a noticeable latency penalty due to the CPU→GPU communication overhead, rendering CPU-GPU to perform poorly than CPU-only. Centaur on the other hand achieves $1.7-17.2\times$ performance speedup and $1.7-19.5\times$ energy-efficiency improvement than CPU-only.

TABLE IV
POWER CONSUMPTION.

	CPU-only	CPU-GPU	Centaur
Power (Watts)	80	91/56 (CPU/GPU)	74

VII. DISCUSSION

Given the limited availability of chiplet-based, package integrated CPU+FPGA devices, we utilized Intel HARPv2 as a proof-of-concept substrate to demonstrate the merits of Centaur. With recent advances in packaging (e.g., Intel EMIB [43] and Foveros [31]) and package-level signaling technologies (e.g., NVIDIA's ground-referenced signaling [6], [57]), architects are provided with rich a set of tools for designing chiplet-based CPU+FPGA architectures. This section discusses some key design paramaters of CPU+FPGAs and its implication in designing accelerators for recommendations.

CPU↔FPGA bandwidth. While our baseline CPU+FPGA platform provides *only* 28.8 GB/sec of CPU↔FPGA unidirectional communication bandwidth, upcoming package-level signaling technologies are expected to deliver several hundreds of GB/sec of communication throughput across chiplets [6], [57]. As discussed in Section III, the limited parallelism and throughput in gathering embedding vectors is one of the key obstacles for CPU-only designs. Note that embedding gather operations are inherently a collective operation where all embedding vectors must be gathered first in order to proceed to the following feature interaction stage. Because a significant fraction of vector reads are cache misses however, the gathering embeddings suffer from significant latency overheads due to the implicit barrier enforced in gathers. An interesting CPU+FPGA design point is to optimize the overall architecture for throughput, rather than locality, and allow the high-bandwidth FPGA→CPU embedding vector read operations to bypass the CPU cache hierarcy (as discussed in Section IV-B, Figure 8), maximizing available parallelism and throughput. Care must be taken however to guarantee cache coherence and consistency, which require carefully co-designed cache primitives for sparse embedding layers. Exploring such design point is part of our next future work.

FPGA size. State-of-the-art FPGA-based dense accelerators provide several tera-operations scale of throughput, thanks to the abundant reconfigurable logic units available within the latest FPGA device (e.g., Cloud-DNN provides 1.8 TOPS of throughput over a Xilinx VU9P board [12]). Given the embarrassingly parallel nature of DNN algorithms, we expect the effective throughput of our dense accelerator to proportionally scale up once the latest FPGA technology is integrated with the CPU. This must of course be accompanied by a high-throughput CPU↔FPGA communication channel

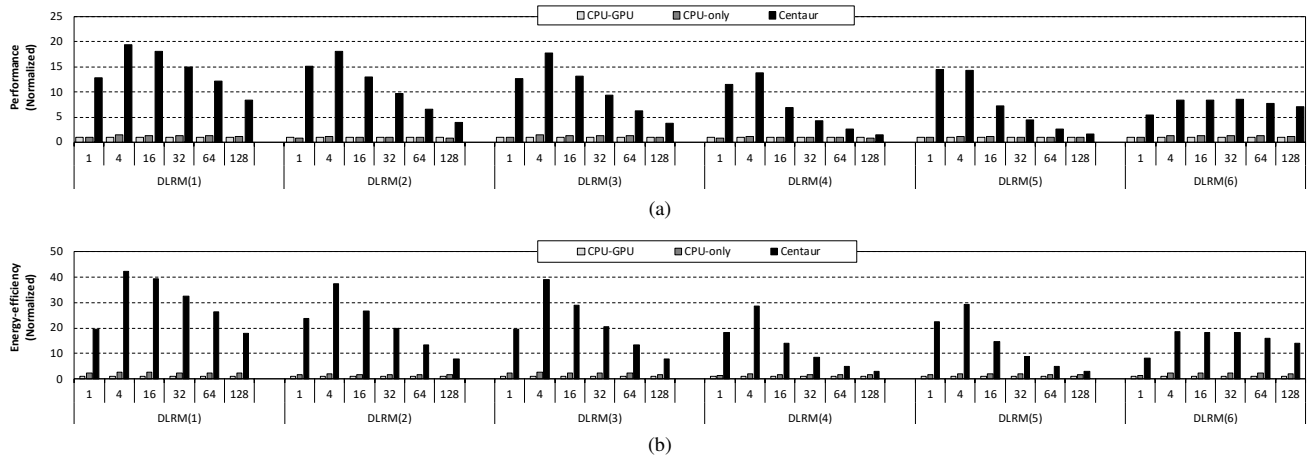

Fig. 15. Centaur's (a) performance and (b) energy-efficiency improvement compared to CPU-only and CPU-GPU. All results are normalized to CPU-GPU which exhibits the lowest performance and energy-efficiency.

across chiplets in order to to proportionally feed enough input tensors to the accelerator, which can be delivered using the aforementioned, high-speed/high-bandwidth package-level signaling technology.

VIII. Related Work

Recommendation models are the backbone ML algorithm that supports a variety of internet services thus having significant industrial importance. While several hyperscalers [18], [23], [23], [25], [27], [54] hint at the scale of compute and memory required to deploy recommendations, little attention has been paid from the computer systems community to address this important research space (e.g., Wu et al. [63] states that only 2.1% of research papers published in top computer architecture venues studies recommendation models). Our recent work on TensorDIMM [38] was one of those few earlier works [22], [36], [38] in the architecture community to explore this research area, proposing a hardware/software co-design for embedding layers. TensorDIMM employs a DIMM-based near-memory processing unit [2], [7] in a disaggregated GPU memory system as means to overcome the memory bandwidth bottlenecks of embedding layers. While the problem space Kwon et al. tackles is identical to Centaur, the following factors render our study unique compared to TensorDIMM. First, the focus of our work is on *CPU-centric* systems which is the most commonly adopted inference deployment setting by hyperscalers, unlike TensorDIMM which assumes a *GPU-centric* system for inference. Second, TensorDIMM requires a separate, pooled memory architecture to achieve maximum performance benefits, which impacts the overall compute density of the overall datacenter, potentially impacting TCO. Centaur has been carefully designed from the ground up to be minimally intrusive to existing server nodes as our chiplet-based CPU+FPGA based solution is socket-compatible to current systems, easing its adoption. Third, TensorDIMM is based on a near-memory processing paradigm which requires modifications to the GPU ISA, system software, and the runtime system, unlike Centaur which can be implemented using existing package-integrated CPU+FPGA technology. Lastly, TensorDIMM relies on rank-level parallelism to increase the effective throughput of embedding gather operations, so the benefits of TensorDIMM is limited to sufficiently *wide* embedding vectors, constraining the algorithmic nature of recommendation models. Our solution is not tied to a particular embedding vector size and is hence much more flexible and applicable for a variety of recommendation algorithms. Overall, the key contribution of our work on Centaur is orthogonal to TensorDIMM and stands unique on its own. Table V is a summary of comparison between Centaur and closely related work.

IX. Conclusion

In this paper, we utilize an emerging, package-integrated CPU+FPGA technology to demonstrate an end-to-end acceleration of personalized recommendation models. Our hybrid, sparse-dense Centaur architecture synergistically combines a sparse accelerator for embedding gathers/reductions and a dense accelerator for GEMM computations, holistically addressing the dual challenges of memory bandwidth and compute throughput. Using a prototype implementation of our proposal on Intel HARPv2 device, Centaur achieves $1.7-17.2\times$ and $1.7-19.5\times$ performance and energy-efficiency improvement, respectively, compared to conventional CPU-only systems.

Acknowledgment

This research is supported by Samsung Research Funding Center of Samsung Electronics (SRFC-TB1703-03). We thank Jaewoong Sim and Intel Labs for giving us access to the CPU+FPGA system through the Hardware Accelerator Research Program (HARP). We also thank the anonymous reviewers and Yunjae Lee from our research group for their constructive feedback that helped improve the final version of this paper.

TABLE V
COMPARISON BETWEEN CENTAUR AND PRIOR WORK.

	TABLA [42]	DNNWEAVER [58]	DNNBuilder [67]	Cloud-DNN [12]	Chameleon [7]	TensorDIMM [38]	Ours
Transparent to existing hardware	✓	✓	✓	✓			✓
Transparent to existing software	✓	✓	✓	✓			✓
Work on accelerating dense DNNs	✓	✓	✓	✓			✓
Applicable for accelerating gathers					✓	✓	✓
Applicable for small vector loads					✓		✓
Study on recommendation models						✓	✓

REFERENCES

[1] J. Albericio, P. Judd, T. Hetherington, T. Aamodt, N. E. Jerger, and A. Moshovos, "Cnvlutin: Ineffectual-Neuron-Free Deep Convolutional Neural Network Computing," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[2] M. Alian, S. W. Min, H. Asgharimoghaddam, A. Dhar, D. K. Wang, T. Roewer, A. McPadden, O. O'Halloran, D. Chen, J. Xiong, D. Kim, W. Hwu, and N. S. Kim, "Application-Transparent Near-Memory Processing Architecture with Memory Channel Network," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[3] Altera, "Floating-Point IP Cores User Guide," 2016.

[4] M. Alwani, H. Chen, M. Ferdman, and P. Milder, "Fused Layer CNN Accelerators," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[5] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep Speech 2: End-To-End Speech Recognition in English and Mandarin," 2015.

[6] A. Arunkumar, E. Bolotin, B. Cho, U. Milic, E. Ebrahimi, O. Villa, A. Jaleel, C.-J. Wu, and D. Nellans, "MCM-GPU: Multi-Chip-Module GPUs for Continued Performance Scalability," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[7] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and Practical Near-DRAM Acceleration Architecture for Large Memory Systems," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[8] Caffe2, "Sparse Operations," 2017.

[9] M. Campo, C.-K. Hsieh, M. Nickens, J. Espinoza, A. Taliyan, J. Rieger, J. Ho, and B. Sherick, "Competitive Analysis System for Theatrical Movie Releases Based on Movie Trailer Deep Video Representation," in *arxiv.org*, 2018.

[10] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, 2014.

[11] Y. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[12] Y. Chen, J. He, X. Zhang, C. Hao, and D. Chen, "Cloud-DNN: An Open Framework for Mapping DNN Models to Cloud FPGAs," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[13] Y. Chen, T. Krishna, J. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in *Proceedings of the International Solid State Circuits Conference (ISSCC)*, 2016.

[14] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2014.

[15] Y. Choi and M. Rhu, "PREMA: A Predictive Multi-task Scheduling Algorithm For Preemptible Neural Processing Units," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2020.

[16] J. Choquette, "Volta: Programmability and Performance," in *Hot Chips: A Symposium on High Performance Chips*, 2017.

[17] P. Covington, J. Adams, and E. Sargin, "Deep Neural Networks for Youtube Recommendations," in *Proceedings of the ACM Conference on Recommender Systems (RECSYS)*, 2016.

[18] J. Dean, D. Patterson, and C. Young, "A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution," in *IEEE Micro*, 2018.

[19] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *arxiv.org*, 2018.

[20] C. Gao, D. Neil, E. Ceolini, S.-C. Liu, and T. Delbruck, "DeltaRNN: A Power-efficient Recurrent Neural Network Accelerator," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[21] Google, "Cloud TPUs: ML Accelerators for TensorFlow," 2017.

[22] U. Gupta, S. Hsia, V. Saraph, X. Wang, B. Reagen, G.-Y. Wei, H.-H. S. Lee, D. Brooks, and C.-J. Wu, "DeepRecSys: A System for Optimizing End-To-End At-scale Neural Recommendation Inference," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2020.

[23] U. Gupta, C.-J. Wu, X. Wang, M. Naumov, B. Reagen, D. Brooks, B. Cottel, K. Hazelwood, M. Hempstead, B. Jia, H.-H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The Architectural Implications of Facebook's DNN-based Personalized Recommendation," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2020.

[24] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. J. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[25] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[26] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural Collaborative Filtering," in *Proceedings of the International Conference on World Wide Web (WWW)*, 2017.

[27] J. Hestness, N. Ardalani, and G. Diamos, "Beyond Human-Level Accuracy: Computational Challenges in Deep Learning," in *Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPOPP)*, 2019.

[28] B. Hyun, Y. Kwon, Y. Choi, J. Kim, and M. Rhu, "NeuMMU: Architectural Support for Efficient Address Translations in Neural Processing Units," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)*, 2020.

[29] Intel, "Hardware Accelerator Research Program (HARP)," 2017.

[30] Intel, "Intel Agilex FPGAs and SoCs," 2019.

[31] Intel, "Intel Foveros 3D Packaging Technology," 2019.

[32] Intel, "Intel VTune Profiler," 2020.

[33] H. Jang, J. Kim, J.-E. Jo, J. Lee, and J. Kim, "MnnFast: A Fast and Scalable System Architecture for Memory-Augmented Neural Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2019.

[34] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. luc Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Na-

garajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[35] W. Jung, D. Jung, B. Kim, S. Lee, W. Rhee, and J. Ahn, "Restructuring Batch Normalization to Accelerate CNN Training," in *The Conference on Systems and Machine Learning (SysML)*, 2019.

[36] L. Ke, U. Gupta, C.-J. Wu, B. Y. Cho, M. Hempstead, B. Reagen, X. Zhang, D. Brooks, V. Chandra, U. Diril, A. Firoozshahian, K. Hazelwood, B. Jia, H.-H. S. Lee, M. Li, B. Maher, D. Mudigere, M. Naumov, M. Schatz, M. Smelyanskiy, and X. Wang, "RecNMP: Accelerating Personalized Recommendation with Near-Memory Processing," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2020.

[37] Y. Kwon and M. Rhu, "A Disaggregated Memory System for Deep Learning," in *IEEE Micro*, 2019.

[38] Y. Kwon, Y. Lee, and M. Rhu, "TensorDIMM: A Practical Near-Memory Processing Architecture for Embeddings and Tensor Operations in Deep Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019.

[39] Y. Kwon and M. Rhu, "A Case for Memory-Centric HPC System Architecture for Training Deep Neural Networks," in *IEEE Computer Architecture Letters*, 2018.

[40] Y. Kwon and M. Rhu, "Beyond the Memory Wall: A Case for Memory-Centric HPC System for Deep Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2018.

[41] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, and T. Chen, "Cambricon: An Instruction Set Architecture for Neural Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2016.

[42] D. Mahajan, J. Park, E. Amaro, H. Sharma, A. Yazdanbakhsh, J. K. Kim, and H. Esmaeilzadeh, "TABLA: A Unified Template-based Framework for Accelerating Statistical Machine Learning," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2016.

[43] R. Mahajan, R. Sankman, N. Patel, D. Kim, K. Aygun, Z. Qian, Y. Mekonnen, I. Salama, S. Sharan, D. Iyengar, and D. Mallik, "Embedded Multi-die Interconnect Bridge (EMIB) – A High Density, High Bandwidth Packaging Interconnect," in *IEEE Electronic Components and Technology Conference (ECTC)*, 2016.

[44] D. J. Moss, S. Krishnan, E. Nurvitadhi, P. Ratuszniak, C. Johnson, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "A Customizable Matrix Multiplication Framework for the Intel HARPv2 Xeon+FPGA Platform: A Deep Learning Case Study," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2018.

[45] D. J. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra, and P. H. Leong, "High Performance Binary Neural Networks on the Xeon+FPGA Platform," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[46] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep Learning Recommendation Model for Personalization and Recommendation Systems," in *arxiv.org*, 2019.

[47] N. Nethercote and J. Seward, "Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2007.

[48] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. O. G. Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?" in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[49] NVIDIA, "The NVIDIA DGX-1V Deep Learning System," 2017.

[50] NVIDIA, "NVIDIA Tesla V100," 2018.

[51] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "SCNN: An Accelerator for Compressed-sparse Convolutional Neural Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[52] E. Park, D. Kim, and S. Yoo, "Energy-efficient Neural Network Accelerator Based on Outlier-aware Low-precision Computation," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2018.

[53] J. Park, H. Sharma, D. Mahajan, J. K. Kim, P. Olds, and H. Esmaeilzadeh, "Scale-Out Acceleration for Machine Learning," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2017.

[54] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. Khudia, J. Law, P. Malani, A. Malevich, S. Nadathur, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. H. an Bill Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications," in *arxiv.org*, 2018.

[55] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[56] M. Rhu, M. O'Connor, N. Chatterjee, J. Pool, Y. Kwon, and S. W. Keckler, "Compressing DMA Engine: Leveraging Activation Sparsity for Training Deep Neural Networks," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2018.

[57] Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, "Simba: Scaling Deep-Learning Inference with Multi-Chip-Module-Based Architecture," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2019.

[58] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Misra, and H. Esmaeilzadeh, "From High-Level Deep Neural Models to FPGAs," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016.

[59] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN Accelerator Efficiency Through Resource Partitioning," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[60] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee, "Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba," in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2018.

[61] P. N. Whatmough, S. K. Lee, N. Mulholland, P. Hansen, S. Kodali, D. C. Brooks, and G.-Y. Wei, "DNN ENGINE: A 16nm Sub-uJ Deep Neural Network Inference Accelerator for the Embedded Masses," in *Hot Chips: A Symposium on High Performance Chips*, 2017.

[62] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine Learning at Facebook: Understanding Inference at the Edge," in *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 2019.

[63] C.-J. Wu, D. Brooks, U. Gupta, H.-H. Lee, and K. Hazelwood, "Deep Learning: Its Not All About Recognizing Cats and Dogs," 2019.

[64] Q. Xiao, Y. Liang, L. Lu, S. Yan, and Y. Tai, "Exploring Heterogeneous Algorithms for Accelerating Deep Convolutional Neural Networks on FPGAs," in *Design Automation Conference (DAC)*, 2017.

[65] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015.

[66] J. Zhang and J. Li, "Improving the Performance of OpenCL-based FPGA Accelerator for Convolutional Neural Network," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[67] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. Hwu, and D. Chen, "DNNBuilder: An Automated Tool for Building High-Performance DNN Hardware Accelerators for FPGAs," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2018.