2021 31st International Conference on Field-Programmable Logic and Applications (FPL) | 978-1-6654-3759-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/FPL53798.2021.00057

# Distributed Recommendation Inference on FPGA Clusters

Yu Zhu*, Zhenhao He*, Wenqi Jiang*, Kai Zeng†, Jingren Zhou†, Gustavo Alonso*

*Systems Group, ETH Zurich
yuzhuyu@ethz.ch, {zhe, wenqi.jiang, alonso}@inf.ethz.ch

†Alibaba Group
kaizeng.zk@gmail.com, jingren.zhou@alibaba-inc.com

***Abstract*—Deep neural networks are widely used in personalized recommendation systems. Such models involve two major components: the memory-bound embedding layer and the computation-bound fully-connected layers. Existing solutions are either slow on both stages or only optimize one of them. To implement recommendation inference efficiently in the context of a real deployment, we design and implement an FPGA cluster optimizing the performance of both stages. To remove the memory bottleneck, we take advantage of the High-Bandwidth Memory (HBM) available on the latest FPGAs for highly concurrent embedding table lookups. To match the required DNN computation throughput, we partition the workload across multiple FPGAs interconnected via a 100 Gbps TCP/IP network. Compared to an optimized CPU baseline (16 vCPU, AVX2-enabled) and a one-node FPGA implementation, our system (four-node version) achieves 28.95× and 7.68× speedup in terms of throughput respectively. The proposed system also guarantees a latency of tens of microseconds per single inference, significantly better than CPU and GPU-based systems which take at least milliseconds.**

## I. Introduction

Personalized recommendation systems have widely adopted deep neural networks to improve user experience and business revenue. For example, Facebook feeds social media news using DNN-based recommender models [1]; Google applies both deep and linear models for Youtube and App Store recommendations [2], [3]; and Alibaba employs several DNN architectures for a wide range of e-commerce use cases [4]. Due to its popularity, DNN-based recommendation inference may comprise up to 79% of the machine learning inference workloads in some data centers [1]. As a result, it is crucial to optimize its overall performance both in terms of data center efficiency as well as from an application perspective.

Fig. 1 shows a typical recommendation model for *Click-Through Rate (CTR)* prediction (how likely the user will click on the recommended content), the type of system we target in this paper. Unlike regular DNNs that usually take a dense feature vector as input, recommendation models contain both dense features (e.g., age or gender) as well as sparse features (e.g., place of residence or user account ID). The model regards each sparse feature as an entry index to retrieve a dense embedding vector from an embedding table. The retrieved embedding vectors are then concatenated with the dense features and fed to several fully-connected (FC) layers to predict the CTR.

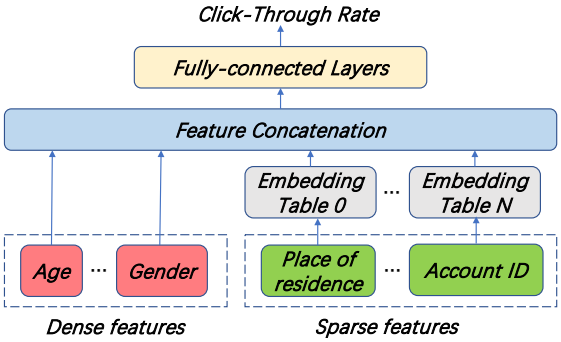


Fig. 1. High-level architecture of a typical deep recommendation model.

The performance of recommendation inference is limited by two main bottlenecks and constrained by strict service-level agreements (SLA). First, the embedding table lookup process significantly bounds the inference performance. These lookup operations induce a massive number of random DRAM access because there are many embedding tables (tens to hundreds), and each lookup only retrieves a short embedding vector (usually 4 to 64 dimensions). Such embedding lookups hamper GPU acceleration. Second, the DNN computation performance (fully-connected layers) is computationally heavy, which limits what can be achieved with CPU-based systems. Finally, for real-time recommendations, a strict SLA of latency of less than tens of milliseconds must be met. As a result, the *Performance Metric* of recommendation is throughput under a given SLA. This has further implications, especially for CPU-based systems, as small batches of inference requests must be used to meet the latency requirements.

The existing FPGA designs for recommendation inference are mainly optimized on one of the bottlenecks rather than both. Centaur [5] is an FPGA accelerator for recommendations based on Intel's Xeon+FPGA platform. The speedup mainly roots in the accelerated FC layer computation: due to the limited number of CPU-side memory channels, the embedding lookup bottleneck still exists. MicroRec [6] takes advantage of the High-Bandwidth Memory (HBM) available on Xilinx's Alveo U280 to enable highly concurrent embedding table lookups. Nevertheless, the computation significantly limits the

978-1-6654-3759-2/21/$31.00 ©2021 IEEE
DOI 10.1109/FPL53798.2021.00057



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

overall performance after the memory bottleneck is eliminated: due to the hardware resource consumption of the lookup unit, the resources left for the computation are limited, thereby slowing down the second part of the inference process. As a result, it is hard to achieve high throughput on both of the stages in recommendations on a single FPGA.

**Our approach**. In this paper, we design and implement an FPGA cluster for recommendation inference to achieve high performance on both the embedding lookups and the FC layer computation while guaranteeing low inference latency. By using an FPGA cluster, we can still place the embedding table lookup module on an FPGA equipped with HBM for high-performance lookups. In the meanwhile, the extra FPGA nodes provide sufficient hardware resources for FC layer computation, such that the throughput of the computation can match embedding table lookups. To enable reliable data transmission, we adopt an open-source 100 Gbps TCP/IP stack tailored for Vitis HLS [27]. Interconnecting FPGAs via the network enables (a) low transmission with a latency within several microseconds and (b) convenient deployment of the proposed system in data centers where FPGAs mainly communicate through the network [8]–[10].

**Key results**. To illustrate the effectiveness of our system, we conduct experiments on a model used in production provided by an *anonymous company* and evaluate the performance of the proposed system using 2 and 4 FPGAs. Compared with the CPU baseline (16 vCPU; 128 GB DRAM with 8 channels; AVX2-enabled), our system achieves 4.32× and 7.68× speedup in terms of throughput and an inference latency of 12.06 and 20.05 microseconds. Because of (a) the relatively balanced performance on the embedding layer and the fully-connected layers, and (b) the low-latency and high-throughput network interconnection, our design also shows clear advantages over a single FPGA implementation: the throughput per node in the FPGA cluster is 2.16× and 1.92× that of a single FPGA, and the inference latency is also slightly better thanks to the more hardware resources assigned for FC layer computation and the low network latency.

**Contributions**. Our contributions in this paper include:

- We show how to build an FPGA cluster interconnected by 100 Gbps TCP/IP network for recommendation inference.
- We explore how to distribute the FC layers across several FPGA nodes so that the computation performance is comparable to the high-performance embedding lookup enabled by HBM.
- We evaluate the system's performance on a real-world recommendation model. It achieves 16.31∼28.95× throughput over CPU and an inference latency of 12.06∼20.05 microseconds.

## II. Background & Motivation

### A. Deep Recommendation Model

Fig. 1 shows the recommendation model we aim to accelerate. The input of the model contains both dense and sparse features. During inference, the recommendation system first translates each sparse feature to an entry index to retrieve a dense vector from the corresponding embedding table. Then, the retrieved vectors are concatenated with dense features before they are fed to several fully-connected layers for CTR prediction, i.e., calculating the probability the user will click on the recommended item. Table I lists the parameters of the model used in our experiment. The model extracts embedding vectors from 98 embedding tables, concatenates them into an 876-dimensional dense vector and predicts CTR through a 3-layer DNN. Although there are alternative DNN architecture designs [1], [2], [4], [11], [12], these models are still composed of the two major building blocks, i.e., the embedding tables and the DNN classifier, thus sharing the inference bottlenecks we describe below.

TABLE I
Parameters of the experimental recommendation model.

| Table Num | Concat Feat Length | Hidden Layers | Size |
|---|---|---|---|
| 98 | 876 | (1024, 512, 256) | 15.1GB |

### B. Bottlenecks and Challenges

There are two performance bottlenecks in recommendation inference. While the first bottleneck is DNN computation as in many other inference workloads, looking up embedding vectors is a unique bottleneck in recommendation inference. This is because each embedding vector is short (usually containing between 4 and 64 elements) while the number of tables is large (from tens to hundreds). Retrieving and gathering short vectors from different memory addresses then leads to many random DRAM accesses and low embedding lookup performance.

Aside of these two performance bottlenecks, a recommendation inference system must meet strict latency constraints within tens of milliseconds in order to provide real-time recommendation services. As a result, the *performance metric* of recommendation inference is inference throughput under SLA. Although large batch sizes are used for many inference tasks to improve throughput for CPU and GPU-based systems, the batch sizes are bound by the latency constraint as the larger the batch, the higher the latency. Both the restricted throughput due to using small batches and the irregular embedding lookup operations hamper the use of GPUs as recommendation inference accelerators in data centers.

To demonstrate the limitations of CPU-based deployments, Fig. 2 shows the performance profile of recommendation inference on a production model. We conduct the experiment on a CPU server (16 vCPU; 128 GB DRAM; AVX2 enabled) and use batch sizes that can meet the SLA. Even when using a relatively large batch size (1024) given the latency constraint, computation and embedding table lookup have a similar share of the inference latency. Thus, to implement a high-performance recommendation inference system, it is important to significantly speedup both the embedding lookups as well as the DNN computation.



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

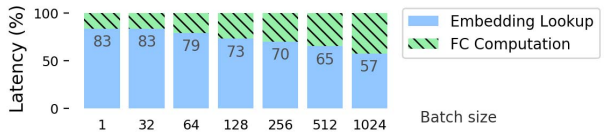


Fig. 2. Apart from DNN computation, recommendation inference in CPU-based systems is also significantly bound by the embedding table lookups.

### C. Existing FPGA Solutions

Although there have been some research exploring FPGA designs for recommendation inference, existing work has failed to eliminate both of these bottlenecks just described (embedding lookups and DNN computation). We summarize these efforts and point out the need for novel FPGA-based recommendation inference systems.

- Centaur [5] is an FPGA accelerator for recommendation inference. It employs an Intel's Xeon+FPGA architecture with shared memory space between the CPU and the FPGA. Although the performance is better than a CPU-based inference engine, the embedding table lookup bottleneck still exists. The reason is that the FPGA needs to access the CPU-side memory to retrieve embedding vectors through the few memory channels available in CPU-based systems. As a result, the speedup is mainly achieved from the fast DNN computation on the FPGA.
- MicroRec [6] eliminates the memory bottleneck by taking advantage of the High-Bandwidth Memory (HBM) recently available on FPGAs (Xilinx Alveo U280). However, the embedding lookup module interconnected with HBM consumes a significant portion of the on-chip resources, leaving fewer resources for the DNN computation. As a result, the computation becomes the bottleneck after removing the memory bottleneck.

In this paper, we aim to design an FPGA-based recommendation inference system that can tackle both the memory and computation bottlenecks, thus achieving a significant speedup compared with existing solutions.

### D. Acceleration with FPGA Clusters

Work has been done exploring efficient DNN inference on FPGAs, both on a single node and on FPGA clusters [13], [14]. Most of the approaches use FPGA clusters connected with point-to-point serial links and proprietary protocols [8], [15]–[17], where the cost to build such a cluster is high and the flexibility of the deployment is limited as the topology is fixed. In contrast, we focus on FPGA clusters embedded on the network data path and connected with the rest of the data center infrastructure (e.g., high bandwidth links and network switches), as indicated by Microsoft Brainwave [18], [19]. In such a platform, FPGAs could be deployed with hardware network stacks using protocols that are commonly available in the data center, such as UDP [20], TCP/IP [21]–[24] and RDMA [25], [26]. For industrial-level recommendation systems that usually run on such data centers, these platforms

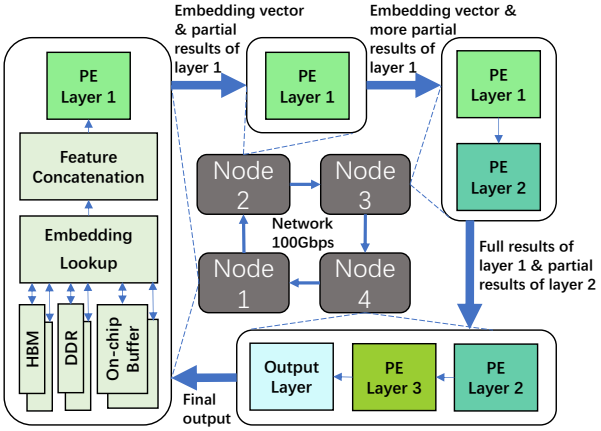


Fig. 3. Overall architecture of distributing the target inference model with a four-node FPGA cluster. Each node is connected to other nodes via up to 100 Gbps network.

are very helpful since the inference of the recommender model, which is one small step in the whole recommendation pipeline, can be offloaded to the FPGA cluster without the need to build a customized network.

## III. Design and Implementation

In this section, we introduce how to design and implement an FPGA cluster for recommendation inference. We start by reviewing the system (Section III-A), then specify the building blocks used for constructing the inference pipeline (Section III-B), and finish by describing the principles to partition the inference workload to multiple FPGA-nodes (Section III-C).

### A. Design Overview

We build an FPGA cluster for recommender model inference. FPGAs are connected to each other via an open-source 100 Gbps hardware network TCP/IP stack [27]. In the FPGA cluster, we use one FPGA to store the embedding tables. The computation of the neural network is distributed to several FPGAs in a deeply pipelined fashion due to the feed-forward property of the inference.

Fig. 3 shows an example of a design for partitioning the model on four FPGAs. The model can also be partitioned to a varying number of FPGAs following the partitioning principles detailed in section III-C. In such a design, one FPGA takes advantage of the FPGA HBM, DDR and on-chip memory to store the embedding tables and serves recommendation system inference queries. An embedding lookup module accesses the embedding tables in parallel, and then all small feature vectors are concatenated together to form the embedding vector. The computation of the neural network is mainly performed by numerous processing elements (PE), which are distributed across several nodes. The total number of FPGAs assigned to a neural network layer is related to the computation requirements of each layer. The first layer is distributed across three nodes in our design, including the node with the embedding table,



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

since it requires the most computation power. The second layer is distributed across two nodes and the third layer only requires part of a node since it consumes the least amount of computation. The output layer operates on the output vector from the third layer to a 32-bit result and finishes the process for inference. Then the last node forwards the final result back to the first node to complete a query. In such a setting, each node relies on the hardware TCP/IP stack to pass embedding vectors or the output of each layer.

### B. Building Blocks

There are two major building blocks in an FPGA accelerator for recommendation inference: the embedding lookup module for embedding vector retrieval and PE for FC layer computation. We show how they are designed in this section and how to build multi-node FPGA systems composed of these building blocks later on.

*1) Embedding Lookup Module:* As suggested by related work [6], we take advantage of the hybrid memory resources available on the Xilinx's Alveo U280 FPGA to enable highly concurrent embedding table lookups. The embedding table sizes are fairly unbalanced in real-world models, from KBs to GBs, because the entry numbers needed to encode different sparse features vary (e.g., there could be millions of users while there are only hundreds of countries in the world). As a result, we allocate tables to different types of memory according to their sizes. The smallest tables are stored in on-chip memory including BRAM and URAM, while the few largest tables are allocated to the two DDR memory channels. The remaining medium-sized tables are then assigned to the 32 HBM channels. We balance the number of tables stored in every off-chip memory channels (DDR and HBM) so that the embedding lookup workload is balanced for each of them.[1]

The lookup process is highly concurrent. The embedding lookup module dispatches the sparse features as the lookup indexes for respective memory channels. The embedding vectors are then retrieved in parallel and concatenated before they are sent to the FC layer computation module. Due to the high embedding lookup throughput brought by the high retrieval concurrency, the computation becomes the bottleneck if only one FPGA is used to implement both the embedding lookups and the computation [6]. As a result, we partition the computation workload to multiple FPGA nodes while assigning only one node for embedding lookups so as to balance the performance of the two components.

*2) DNN Computation Pipeline:* Fig. 4 shows the computation flow for a single fully-connected layer. The PEs are the basic building blocks for general matrix-matrix multiplications (GEMM). Each PE holds a portion of the weights of a fully-connected layer and is responsible for a partial matrix-matrix multiplication through parallelized multiplications followed by an add tree [28]. During inference, the input feature vectors of the layer are broadcasted to all PEs, and the partial results

[1]The memory controller generated by Vitis exhibits close access latency to DDR and HBM.

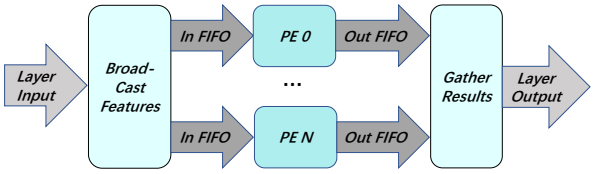


Fig. 4. The parallelized computation flow in a single fully-connected layer.

are gathered after the computation and forwarded to the next layer. Although an alternative interconnects topology, i.e., a systolic array [29], can be used to connect these PEs, the data propagation scheme in the systolic arrays increases the inference latency especially when the number of PE is large. In this work, we aim to achieve ultra-low latency for real-time recommendation serving, thus choose a broadcasting and gathering design for the PE interconnection.

### C. Partitioning the Computation

Partitioning the computation to an FPGA cluster needs to follow some design principles and considering some constraints. We show an example of how to partition our target model to an FPGA cluster containing 4 U280 FPGAs.

*1) Balance Resource Utilization:* We aim to achieve a balanced design in resource utilization across FPGAs since different components in the inference pipeline require different resources. The design of a single FPGA implementation, where the embedding lookup and the neural network inference are performed in the same FPGA device, has an unbalanced resource utilization. In such a design, because embedding tables and the weights of FC layers require huge amount of on-chip memory, the resource bottleneck is the BRAM, leading to inefficient use of DSPs. By partitioning the inference to multiple nodes, we could distribute the weights of FC layers to multiple nodes, relieving the stress of BRAM requirement on the node that stores embedding tables. With more PEs partitioned evenly across nodes, we expect to achieve a better overall performance by increasing the DSPs' utilization rather than being limited by the BRAM usage.

In both 2-node or 4-node implementations, each PE consumes 18 DSPs. The difference is that we can control the computation loop count in PE, for one or more columns. In our four-node cluster design, one U280 FPGA board contains 9024 DSPs and this gives us a maximum of 2004 PEs with 4 FPGAs. To achieve a balanced design, we also need to consider the computational requirements of each layer since the minimum throughput of all layers determines the overall inference throughput. Each layer requires a matrix multiplication of $876\times1024$, $1024\times512$, and $512\times256$, respectively. We round the ratio of the computational requirement of each layer to power of two and this yields 8:4:1. The PE numbers of each layer should be proportional to this ratio and we assign 1024, 512 and 128 PEs for the three layers respectively. We distribute these PEs evenly across 4 nodes and the total number of PEs deployed on each node consumes about 83% of the total DSPs available.



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

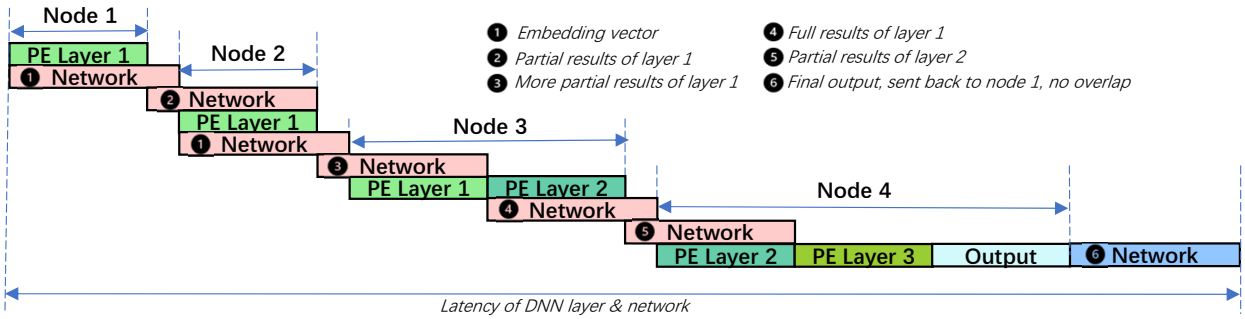


Fig. 5. Overlapping computation and network in a pipelined and parallelized method.

*2) Overlap of Computation & Communication:* Fig. 5 shows how the computation and the network are overlapped among 4 FPGAs, which is crucial to achieving low latency. For each partial layer on different nodes, it requires the same input vector. The network latency of forwarding the input vector to the next partial layer in another node can be overlapped with the computation of the current partial layer. For instance, embedding vectors generated in node 1 are sent to node 2 via network while at the same time, node 1 also performs computation over these embedding vectors. Once the node 1 has partial results, it sends them to node 2 and this process is also partly overlapped with the computation of node 2.

*3) Satisfy Network Bandwidth:* The required network bandwidth of this distributed design should not exceed the actual network bandwidth. For instance, in the design depicted in Fig. 3, the second node requires the most network load since it needs to both receive and send the embedding vector and the intermediate results of the first layer for a single inference. Thus, to make sure the whole design is not bounded by the network, we need to carefully adjust the network bandwidth required by the second node.

## IV. Evaluation

We evaluate the performance of the FPGA cluster for recommendation inference in terms of throughput and latency. We deploy the industrial recommendation model detailed in Table I. Our FPGA solution has a significant advantage in both throughput and latency over the CPU baseline. Compared with a single FPGA implementation, our design achieves a 4.32× and 7.68× speedup on 2 nodes and 4 nodes, respectively, and has lower latency.

### A. Experiment Setup

We run the experiment on an FPGA cluster containing 4 Xilinx Alveo U280 FPGAs, each of which is equipped with 8GB HBM (32 channels) and 32GB DDR4 (2 channels). Each node is connected to each other through a 100 Gbps Cisco Nexus 9336C-FX2 network switch. Vivado HLS is used to translate C++ code to hardware description language (HDL) and program the FPGA for host-kernel interaction. We evaluate our design in two configurations using 2 and 4 FPGAs. For the single node baseline, we refer to MicroRec [6], a design optimized for the FPGA HBMs for parallel lookups and performs DNN computation on a single device. For all FPGA implementations, we have a clocking frequency of 135 MHz except in node 1 in a two-node design, where we have to lower the frequency to 115MHz due to the high resource utilization.

The CPU baseline is tested on an AWS server with Intel Xeon E5-2686 v4 CPU@2.30GHz(16 vCPU, SIMD operations, i.e., AVX2 FMA, supported) and 128GB DRAM(8 channels). We apply an open-source solution on deep recommendation systems [30], where *TensorFlow Serving* [31] supports a highly optimized model inference.

### B. Throughput

Fig. 6 shows a significant speedup in our design in terms of throughput in FPGA clusters over both a single FPGA and the CPU-based system. The throughput of the CPU-based system increases with the batch size but it can not go beyond a batch size of 1024 since it would break the inference latency constraint imposed by the SLA. Therefore, we pick a batch size of 1024 and compare against it. In a single FPGA implementation, by taking advantage of the FPGA HBM for parallel embedding lookup and DSPs for FC layer computation, the system can already achieve a 3.77× speedup compared to CPU baseline. By using 4 FPGAs, we achieve about 28.95× speedup compared to the CPU baseline. Compared with a single FPGA implementation, our design achieves a 2.16× and 1.92× speedup per FPGA node on 2 nodes and 4 nodes, respectively, due to better utilization of resources. Fig. 6 also shows the throughput of the embedding layer (dotted line to the right) for FPGA based solutions. By using 4 FPGAs, we achieve a balanced design where the computation is performed at almost the same rate as the embedding lookup rate.

### C. Latency

Our solution has a significant advantage in terms of latency compared to CPU-based system, as shown in Fig. 7. The CPU-based system has a latency ranging from 7.48 (batch size=1) to 31.72 (batch size=1024) milliseconds. In such a system, the embedding layer is the bottleneck. In contrast, an FPGA-based design has a latency of less than 31.72 microseconds,



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

TABLE II
HARDWARE RESOURCE UTILIZATION(%) IN DIFFERENT FPGA DESIGNS FROM HLS SYNTHESIS REPORT

| | Single | Cluster 2 Nodes | | Cluster 4 Nodes | | | |
|---|---|---|---|---|---|---|---|
| | | Node 1 | Node 2 | Node 1 | Node 2 | Node 3 | Node 4 |
| BRAM | 2516 (62.40%) | 2465 (61.14%) | 1686 (41.82%) | 2122 (52.63%) | 1557 (38.62%) | 1193 (29.59%) | 1933 (47.94%) |
| URAM | 770 (80.21%) | 446 (46.46%) | 444 (46.25%) | 414 (43.13%) | 414 (43.13%) | 444 (46.25%) | 510 (53.13%) |
| DSP | 5193 (57.55%) | 8064 (89.36%) | 8073 (89.46%) | 7488 (82.98%) | 7488 (82.98%) | 7488 (82.98%) | 7497 (83.08%) |
| LUT | 452804 (34.73%) | 559568 (42.92%) | 500131 (38.36%) | 521302 (39.99%) | 461061 (35.37%) | 463638 (35.56%) | 506824 (38.88%) |

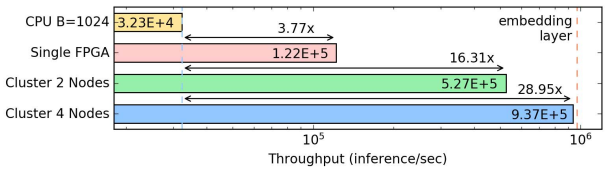


Fig. 6. Comparison of CPU-based system with batch size 1024, single FPGA design and FPGA cluster with 2 or 4 nodes in terms of throughput for inference. The red dotted line is the throughput of embedding layer.

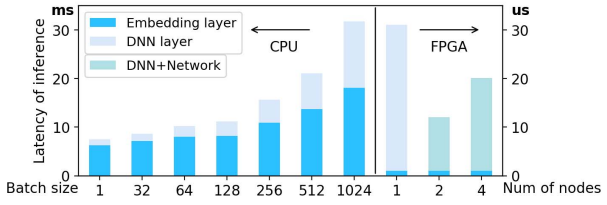


Fig. 7. Comparison in terms of latency in CPU-based system with different batch sizes and different FPGA designs. The FPGA works in a highly pipelined fashion, thus has a batch size of 1.

orders of magnitude lower than the CPU system. Thanks to the low latency hardware network stack and the data center infrastructure, the latency reduction due to partitioning of the computation out-weights the latency introduced by the network, such that our design with 4 nodes has lower latency than a single FPGA implementation.

### D. Resource Utilization

Table II shows the resource utilization for each configuration. For a single FPGA, the usage of BRAM and URAM is high due to the storage and embedding tables while the DSP usage is less than 60 percent. By partitioning the design across multiple nodes, we achieve more than 80 percent DSP usage in every node while keeping the resource utilization of the on-chip memory to a reasonable level.

## V. RELATED WORK

To the best of our knowledge, this is the first paper accelerating recommendation inference on an FPGA cluster.

*DNN for personalized recommendation*. Neural Collaborative Filtering introduces the embedding layer to DNN-based recommendation models in order to encode the sparse features more efficiently [12]. Google then proposed to add a linear model in addition to the deep model to offer more reliable predictions and deployed such models in YouTube and the App Store [2], [3], [11]. Facebook's social media recommender models include extra fully-connected layers to process the dense input features before they are concatenated with the retrieved embedding vectors [1]. Alibaba employs the attention mechanism on the DNN layer in order to interpret time serial information more efficiently in their e-commerce platforms [4].

*Hardware solutions for recommendation inference*. According to Facebook, deep recommendation inference comprises up to 79% of AI inference cycles in their data centers [1]. Thus, a range of efforts has been invested in designing specialized hardware for high-performance recommendation inference. TensorDIMM [32] resorts to a specialized DRAM micro-architecture by increasing the memory access parallelism and adding near memory processing units for simple gathering operations. RecNMP [33] extends the idea of near-DRAM-processing by supporting more tensor operators in DRAM. DeepRecSys [34] evaluates the performance of GPUs in recommendation systems, but the inference throughput is limited because small batches must be used to meet the strict latency constraints. Centaur [5] implements an FPGA accelerator for recommendation using Intel's Xeon+FPGA platform and achieves significant speedup on FC layer computations. MicroRec [6] eliminates the memory bottleneck by taking advantage of the HBM available on Xilinx's U280 FPGA, but the computation becomes a significant bottleneck afterwards. FleetRec [7] expands this idea by resorting to hybrid GPU-FPGA clusters, in which FPGAs with HBM are responsible for embedding lookup and computation are offloaded to GPUs.

*Distributed FPGA systems*. Zhang [13] presents a multi-FPGA system to accelerate CNNs. By carefully partitioning the workload, it achieves linear speedup over a single FPGA and outperforms multi-core CPUs and GPUs significantly. Owaida [35] develops a datacenter-scale FPGA cluster to support high-performance decision tree ensemble inference. Super-LIP [36] implements an FPGA cluster to accelerate DNN inference and achieves super-linear speedup in terms of throughput against a single FPGA design while reducing the overall inference latency.

## ACKNOWLEDGEMENTS

Part of the work of Wenqi Jiang and Zhenhao He has been funded by the Alibaba Group. We would like to thank Xilinx for their generous donation of the XACC FPGA cluster at ETH Zurich on which the experiments were conducted.



Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.

## REFERENCES


[1] Gupta, U., et al. (2020). The architectural implications of facebook's dnn-based personalized recommendation. IEEE International Symposium on High Performance Computer Architecture (HPCA), IEEE.

[2] Covington, P., et al. (2016). Deep neural networks for youtube recommendations. Proceedings of the 10th ACM Conference on Recommender Systems, ACM.

[3] Zhao, Z., et al. (2019). Recommending what video to watch next. Proceedings of the 13th ACM Conference on Recommender Systems, ACM.

[4] Zhou, G., et al. (2019). Deep interest evolution network for click-through rate prediction. Proceedings of the AAAI Conference on Artificial Intelligence.

[5] Hwang, R., et al. (2020). Centaur: A chiplet-based, hybrid sparse-dense accelerator for personalized recommendations. ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), IEEE.

[6] Jiang, W., et al. (2021). MicroRec: efficient recommendation inference by hardware and data structure solutions. Proceedings of Machine Learning and Systems (MLSys).

[7] Jiang, W., et al. (2021). FleetRec: Large-Scale Recommendation Inference on Hybrid GPU-FPGA Clusters. Proceedings of the 27th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), ACM.

[8] Putnam, A., et al. (2014). A reconfigurable fabric for accelerating large-scale datacenter services. ACM/IEEE 41st International Symposium on Computer Architecture (ISCA), IEEE.

[9] Caulfield, A. M., et al. (2016). A cloud-scale acceleration architecture. 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), IEEE.

[10] Firestone, D., et al. (2018). Azure accelerated networking: Smartnics in the public cloud. 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18).

[11] Cheng, H.-T., et al. (2016). Wide & deep learning for recommender systems. Proceedings of the 1st workshop on deep learning for recommender systems.

[12] He, X., et al. (2017). Neural collaborative filtering. Proceedings of the 26th international conference on world wide web.

[13] Zhang, C., et al. (2016). Energy-efficient cnn implementation on a deeply pipelined fpga cluster. Proceedings of the 2016 International Symposium on Low Power Electronics and Design, ACM.

[14] Umuroglu, Y., et al. (2017) Finn: A framework for fast, scalable binarized neural network inference. Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays.

[15] Jun, S.-W., et al. (2015). A transport-layer network for distributed fpga platforms. 25th International Conference on Field Programmable Logic and Applications (FPL), IEEE.

[16] Bunker, T. and S. Swanson (2013). Latency-optimized networks for clustering fpgas. IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines, IEEE.

[17] Mencer, O., et al. (2009). Cube: A 512-fpga cluster. 5th Southern Conference on Programmable Logic (SPL), IEEE.

[18] Chung, E., et al. (2018). Serving dnns in real time at datacenter scale with project brainwave. IEEE Micro.

[19] Fowers, J., et al. (2018). A configurable cloud-scale dnn processor for real-time AI. ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), IEEE.

[20] "Xup vitis network example (vnx)." [Online]. Available: https://github.com/Xilinx/xupvitisnetworkexample.

[21] Ding, L., et al. (2016). Hardware tcp offload engine based on 10-gbps ethernet for low-latency network communication. International Conference on Field-Programmable Technology (FPT), IEEE.

[22] Ji, Y. and Q.-S. Hu (2011). 40gbps multi-connection tcp/ip offload engine. International Conference on Wireless Communications and Signal Processing (WCSP), IEEE.

[23] Sidler, D., et al. (2015). Scalable 10gbps tcp/ip stack architecture for reconfigurable hardware. IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, IEEE.

[24] Ruiz, M., et al. (2019). Limago: An fpga-based open-source 100 gbe tcp/ip stack. 29th International Conference on Field Programmable Logic and Applications (FPL), IEEE.

[25] Li, B., et al. (2017). Kv-direct: High-performance in-memory key-value store with programmable nic. Proceedings of the 26th Symposium on Operating Systems Principles.

[26] Sidler, D., et al. (2020). StRoM: smart remote memory. Proceedings of the Fifteenth European Conference on Computer Systems.

[27] He, Z., et al. (2021). EasyNet: 100 Gbps Network for HLS. 31th International Conference on Field Programmable Logic and Applications (FPL), IEEE.

[28] Chen, Y., et al. (2014). Dadiannao: A machine-learning supercomputer. 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE.

[29] Wei, X., et al. (2017). Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. Proceedings of the 54th Annual Design Automation Conference.

[30] "Lapis-hong/widedeep." [Online]. Available: https://github.com/Lapis-Hong/wide_deep.

[31] Olston, C., et al. (2017). Tensorflow-serving: Flexible, high-performance ml serving. arXiv preprint arXiv:1712.06139.

[32] Kwon, Y., et al. (2019). Tensordimm: A practical near-memory processing architecture for embeddings and tensor operations in deep learning. Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture.

[33] Ke, L., et al. (2020). RecNMP: Accelerating personalized recommendation with near-memory processing. ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), IEEE.

[34] Gupta, U., et al. (2020). DeepRecSys: A system for optimizing end-to-end at-scale neural recommendation inference. ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), IEEE.

[35] Owaida, M. and G. Alonso (2018). Application partitioning on fpga clusters: Inference over decision tree ensembles. 28th International Conference on Field Programmable Logic and Applications (FPL), IEEE.

[36] Jiang, W., et al. (2019). Achieving super-linear speedup across multi-fpga for real-time dnn inference. ACM Transactions on Embedded Computing Systems (TECS).




Authorized licensed use limited to: ETH BIBLIOTHEK ZURICH. Downloaded on February 25,2022 at 15:31:50 UTC from IEEE Xplore. Restrictions apply.