A Framework for Neural Network Inference on FPGA-Centric SmartNICs

Anqi Guo*‡, Tong Geng†‡, Yongan Zhang‡§, Pouya Haghi*, Chunshu Wu*, Cheng Tan¶, Yingyan Lin§, Ang Li‡,and Martin Herbordt*
*ECE Department, Boston University, Boston, MA
†ECE Department, University of Rochester, Rochester, NY
‡Pacific Northwest National Laboratory, Richland, WA
§Rice University, Houston, TX
¶Microsoft, Redmond, WA
Email: *{anqiguo,haghi,happycwu,herbordt}@bu.edu †{tong.geng}@rochester.edu
‡{anqi.guo,tong.geng,yongan.zhang,ang.li}@pnnl.gov §{yz87,yingyan.lin}@rice.edu ¶{chengtan}@microsoft.com

***Abstract*—FPGA-based SmartNICs offer great potential to significantly improve the performance of high-performance computing and warehouse data processing by tightly coupling support for reconfigurable data-intensive computation with cross-node communication, thereby mitigating the von Neumann bottleneck. Existing work, however, has generally been limited in that it assumes an accelerator model where kernels are offloaded to SmartNICs with most control tasks left to the CPUs. This leads to frequent waiting, reduced performance, and scaling challenges.**

In this work, we propose a new distributive data-centric computing framework, named FCsN, for reconfigurable SmartNIC-based systems. Through a lightweight task circulation execution model and its implementation architecture, FCsN allows the complete detaching of NN kernel execution, control logic, system scheduling, and network communication to the SmartNICs. This boosts performance by: (i) avoiding control dependency with CPUs and (ii) supporting streaming NN kernel execution and network communication at line rate and in a very fine-grained manner. We demonstrate the efficiency and flexibility of FCsN using various types of neural network kernels and applications including deep neural networks (DNN) and graph neural networks (GNN); as these last are both irregular and data intensive they offer an especially robust demonstration. Evaluations using commonly-used neural network models and graph datasets show that a system with FCsN can achieve 10× speedups over the MPI-based standard CPU baselines.

I. Introduction

Network communication is increasingly becoming the performance bottleneck for scaled-out HPC and warehouse applications, as enormous amounts of CPU cycles are devoted to packet processing, contributing to long per-packet latency (Figure 1(a)). To reduce this latency, advanced network interface cards known as SmartNICs have been introduced to handle networking functions such as TCP Segmentation [1] and Generic Receive [2]. In cloud computing, SmartNICs support SR-IOV that forwards packets directly to the Virtual Machine bypassing the hypervisor (Figure 1(b)) [3] [4].

Lately it has been found that if an FPGA can be integrated into the NIC, not only more complex network protocols, but also some data-intensive computation can be efficiently realized when processing network packets, often at line-rate, and without introducing significant overhead (Figure 1(c)) [5]–[7]. With high-bandwidth and low-latency access to network data through Multi-Gigabit Transceivers (MGTs), and programming logic with embedded hard-cores, FPGA-based SmartNICs can be viewed as network-focused streaming-processing accelerators, in addition to network support devices. This is particularly useful for domain-specific computations, such as in machine learning and streaming data analytics, as the FPGAs can be reconfigured as customized accelerators.

Nevertheless, existing FPGA-based SmartNICs are constrained by three limitations. (i) *Host-control*: Although the offloading of some simple compute kernels has been demonstrated, this work generally assumes a host-device programming model, leaving the majority of control, scheduling, and management tasks to the host CPUs. This not only incurs an extra burden on the host CPUs, but also leads to poor utilization of the SmartNICs for handling the control-dependencies with the host through PCIe and software stacks. (ii) *Limited scalability*. Existing SmartNIC applications rarely involve offload of non-local tasks, missing opportunities for system-level designs that can span a distributed cluster, eliminate unnecessary data-movement, and support more efficient scheduling and workload balance. (iii) *Programmability*. As the control is performed by the host, most existing SmartNICs only handle relatively simple kernels. Little support is offered to either system or application developers for designing flexible domain-specific acceleration solutions.

In this work, we address these problems by presenting a user-friendly framework for neural network inference on **F**PGA-**C**entric **s**mart**N**IC (**FCsN**) that can perform computation, communication, and control altogether at the same time, allowing flexible and fine-grained task creation, distribution, execution, and finalization across multiple SmartNIC devices. This results in maximally hiding the computation latency with network communication for streaming applications at line-rate, and achieving high FPGA utilization and high performance at system level by avoiding CPU intervention (Figure 1(d)). Figure 2 illustrates the design stack of FCsN. On the software side, FCsN uses a data-centric programming model (Section 3 A) and is equipped with Python-based programming APIs (Section 3 B); on the hardware side, FCsN is equipped with a hardware-based SmartNIC runtime (Section 4 C) to achieve

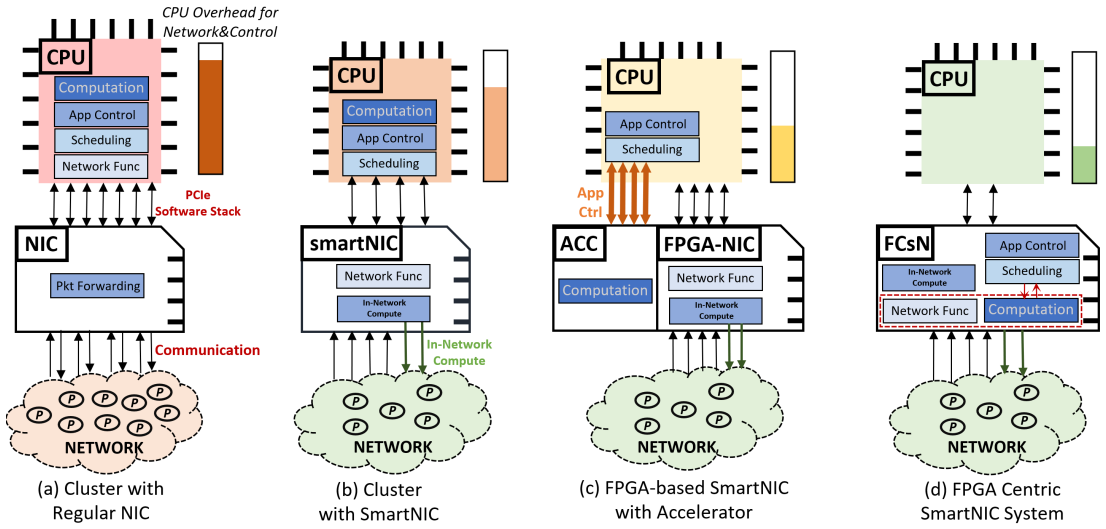

Fig. 1. **(a)** CPU handles computation and network functions; the NIC is under-utilized and the network traffic is heavy resulting in communication bottlenecks. **(b)** SmartNIC handles network functions and performs simple in-network computing. The CPU is in charge of kernel execution. **(c)** FPGA-based SmartNIC acts as a SmartNIC and an accelerator with partially offloaded CPU computations. However, extra overhead between CPU and FPGA-NIC is introduced by CPU's intervention in application control and scheduling. **(d)** FPGA-Centric SmartNIC (FCsN) handles network functions and also application computations, application control, kernel scheduling, and task initiation. CPU cycles are saved, overhead between the NIC and the CPU is reduced, and FPGA-NIC resources are fully utilized.

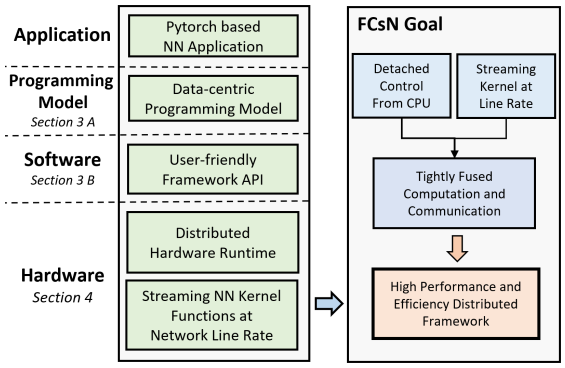

Fig. 2. Overview of FPGA-centric SmartNIC design

CPU-detached scheduling and support high-performance execution of NN kernels at line-rate (Section 4 D). The current FCsN framework focuses on Neural Network applications, but it has the potential of extending to a general framework as many scientific applications share similar basic kernel functions as NN applications. Contributions of this paper are as follows:

- FCsN, a user-friendly and high-performance FPGA-centric SmartNIC framework, which supports domain-specific computation, low-latency communication, and host-detached scheduling;
- A hardware-based FPGA-centric SmartNIC runtime that enables asynchronous and fine-grained task scheduling and so avoids the control dependency with CPUs;
- A series of streaming NN kernels that provide acceleration at line rate and maximally overlap computation latency with network communication for NN applications;
- Evaluations using neural network applications including general DNNs and GNNs with commonly-used models and datasets on systems with FCsN support and realized on Alveo U280 FPGAs. These show that FCsN can achieve 10× speedups over the standard MPI-based system baseline.

II. Background and Related Work

There has been much work utilizing SmartNICs to enhance communication and networking. Dozens of commercial FPGA-based SmartNICs have been released, including from AMD and Intel, e.g., [8]–[10]; surveys include [11], [12]. Other work has focused on near-network processing [13]–[16]. Other FPGA-based network solutions, such as Catapult [17], offload network applications. There has been much work in FPGA-based scalable network stacks supporting TCP/IP, RoCEv2, and UDP/IP [1], [18]–[22]. Work by [23] proposed a configurable network protocol on intelligent NICs. NetFPGA [24] has been invaluable in providing FPGA-based network hardware development environments.

There is also prior art that uses SmartNICs as compute resources [25], [26]. COPA [27] provides a software/hardware framework that makes the underlying FPGA hardware (SmartNIC device) agnostic to middleware. INCA [28] provides general-purpose compute capabilities for SmartNICs that can be utilized when the network is inactive. sPIN [29] provides a portable programming model to offload simple packet processing. NICA [7] provides a framework for inline acceleration of the application data plane on FPGA-based SmartNICs in multi-tenant systems. Other work [2], [30]–[32] supports collectives in FPGA-based hardware.

III. Programming Model & Software Support

In this section, we discuss programming models and introduce FCsN's Python-based programming interface.

A. Programming Models

We investigate two programming models: compute-centric and data-centric. In the compute-centric programming model,

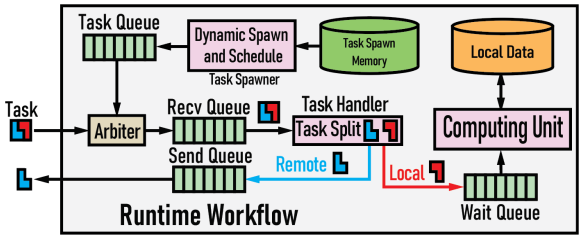

Fig. 3. Data-Centric Workflow

a process is assigned to a processor or an entire node and communication happens through message passing. Computations follow a series of steps: parallel computation (each node participates in a portion of assigned tasks) and barrier synchronization to align execution of nodes. This model works well for BSP applications with easily partitionable data and regular computation. However, many workloads have irregular behavior with skewed data distribution, high synchronization intensity, and irregular communication patterns [33].

The data-centric model [34] is an alternative. It brings computation to the data, rather than the reverse, and so minimizes data movement and reduces unnecessary communication. This model follows a logical ring topology, i.e., the application is partitioned into tasks that circulate around a logical ring system and thereby exploit data locality. Each node can verify whether a task should be executed locally based on the local data range. The data-centric model suits applications with less structured data and irregular behaviors, e.g., involving sparse matrices, that introduce unpredictable data access. For example, modern NN applications, which are ever more optimized to reduce computation, are becoming correspondingly more irregular and communication-bound, especially in large-scale processing [35]. As these are our initial application targets for FCsN, we consider using the data-centric model.

Figure 3 shows the data-centric workflow. Using the data-centric programming model, applications can be split into two parts, data and task. Data is distributed on each node in the system initialization phase; $local_{start}$ and $local_{end}$ indicate the local data range. The RDMA handler fetches data when the task demands remote data. Tasks are pre-registered and dynamically spawned among the nodes. At runtime, tasks circulate among nodes. A task confirms its required data using the starting and ending addresses ($TASK_{start}$ and $TASK_{end}$). Tasks can be replicated for remote nodes if the required data range is wider than the local data range. In this way, the data-centric approach can bring tasks to each node asynchronously rather than sending possibly massive amounts of data through the network.

B. FPGA programming interface

To support a user-friendly interface, a middle layer API coordinates activity between the host CPU and underlying hardware, including system and kernel initialization, hardware status checking, data syncing, and hardware control. A list of API functions with system initialization and finalization is shown in Table I.

FCsN supports most of the major kernels used in Neural Network processing, including 2D convolution, dense and

TABLE I
SOFTWARE PROGRAMMING API AND NEURAL NETWORK KERNELS

Function	Description
Software Programming API	
Overlay_handler = **System_init** (*Node_num*)	FCsN multi-node environment initialization setup
Board_init(*Overlay_handler*, *Bit_file*)	Configure network function, check board status and program the board with binary file
Check_Kernel()	Check kernel status
Data_handler = **Host_data_preprocess** (*data*)	Host preprocess data
Mem_handler = **Board_MEM_init** (*Overlay_handler*, *Data_handler*)	Allocate on-chip Memory, sync and distribute data from host to chip.
Spawn_MEM = **Board_MEM_init** (*Overlay_handler*, *Task_carrying_data*)	Allocate Task Spawner Memory with task required data.
Kernel_Start (*Overlay_handler*, *Task_id*, *Argv*)	Start the kernel with Task_id and control argument
Sync_to_Host(*Overlay_handler*, *Data_handler*)	Sync on-chip Memory data back to host Memory
System_Finalize(*Overlay_handler*)	Free overlay

sparse matrix multiplication, graph aggregation, norm, and non-linear element-wise activation functions. The APIs can be easily and seamlessly integrated into PyTorch-based NN applications. Based on the NN application's need, corresponding kernel function tasks can be configured. Before an application's execution, data that needs to be carried by tasks are synced to on-chip *'Spawn_MEM'*. *'Kernel_start'* starts the task spawner module based on task data from *'Spawn_MEM'* and dynamically generates kernel tasks during runtime based on *'Task_id'*. *'argv'* indicates the task spawner control arguments such as destination or data range required by the task. After starting the kernel, the application is detached from the CPU's control and the hardware runtime handles the control logic. *'Sync_to_Host'* syncs the result back to the host when the application finishes and with *'System_Finalize'* frees the board resources.

To associate the middle layer API with hardware, we use Xilinx Pynq [36] as an API to program and interact with Xilinx XRT [37] and Vitis platform FPGAs. Pynq is an open-source Python-based library for programming both the embedded processors and overlays. We use Vitis HLS to implement basic hardware NN kernel functions. The Vitis compiler compiles HLS kernel into an xo file, creating an overlay by linking with other kernels, e.g. network function and DMA engine.

IV. FPGA-BASED SMARTNIC ARCHITECTURE

We describe the hardware architecture and supported basic NN kernel functions with streaming execution capability.

A. Architecture Overview

The FCsN architecture is shown in Figure 4. The dynamic user logic consists of network function, hardware runtime, and NN kernel compute engines. The network function uses TCP as the Layer 4 transport protocol [1]. Neural network

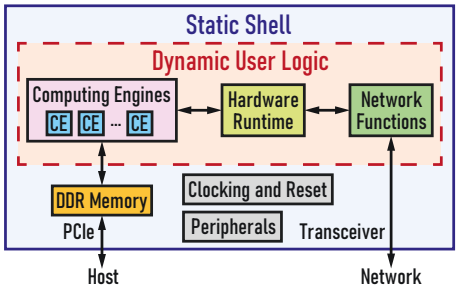

Fig. 4. FPGA-based SmartNIC Overlay

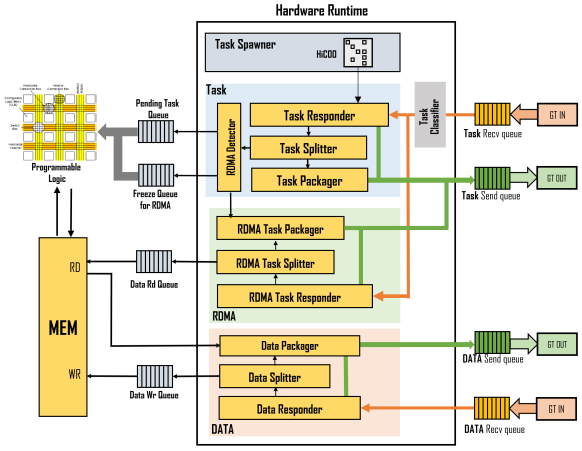

Fig. 5. Hardware Runtime Architecture

kernel functions are configured according to the application's requirement during the initialization phase.

B. Hardware Runtime Support

The hardware runtime is split into two stages, runtime initialization and runtime execution. Runtime initialization enables the host to program the dynamic application chip region with the user's application binary and syncs data from host memory to FPGA's memory. Runtime execution handles application task spawning, scheduling, application control logic, and manages RDMA data.

1) Runtime Initialization: Runtime initialization loads the binary, allocates resources, and prepares for runtime execution. Distributed application kernel tasks are assigned to each node and distributed data is preloaded onto the chip. Runtime execution invokes the corresponding task spawner and dynamically generates tasks with preloaded information.

2) Hardware Runtime: After initialization, the runtime task spawner dynamically generates tasks based on the *'Spawn_MEM'* task info. In this design, each task is packaged as a 512-bit packet with a 32-bit application header and 480-bit payload. The header contains information on task type, task ID, task destination, and task data range. After the hardware runtime has parsed the packet, the task handler consumes the task if current node is within the demanded data range (Figure 3). Packets have three types: Task, RDMA, and Data. Figure 5 shows the three types of runtime handlers.

Task Handler: The Task Classifier pops the first valid packet and appends it to the corresponding handler. Within each handler, the Task Responder checks the integrity of the packet and whether the packet can be fully or partially executed in the current node (Figure 3). (1) Partially executed means the current node does not have all the data it needs. In this case, the Task Splitter splits the packet, consumes the local task and generates a remote packet with the rest of the data range. The Task Packager wraps the remote task and pushes it into the send queue. (2) The consuming task is sent to an RDMA Detector to check if the task needs remote data. (i) If no remote data is needed, it is appended to the pending task queue and assigned to the compute unit. (ii) If the task requests remote data, it is temporarily pushed to a freeze queue and waits for the requested data to be available locally. The RDMA Handler generates a corresponding RDMA task.

RDMA Handler: An RDMA task requests remote data. It is analyzed in the RDMA Task Responder to determine whether the requesting data are available on the local node. After consuming the RDMA task, the task is pushed into a data read queue and waits for the memory controller to issue its requested data. When the data have been retrieved, they are wrapped by the Data Packager and sent to the requesting source node.

Data Handler: handles data packets when the requested data from the current node's RDMA task has been successfully retrieved and informs the RDMA Detector whether there are tasks ready to be compute.

C. Neural Network Kernel Support

In FCsN, non-conflict streaming kernel execution at network line rate leverages the SmartNIC capabilities. Software-hardware co-design of conflict-free kernel functions on the FPGA-based SmartNIC guarantees that kernel execution is not stalled and can be tightly combined with the network pipeline. We provide kernel functions for Neural Network applications including 2D Convolution, Matrix Multiplication, Function Norms, Non-linear element-wise Activation, and Aggregation.

With the aim of achieving stream execution at line rate, the compute pipeline is required to have the capability of consuming one task packet each cycle. Within the pipeline, sub-tasks in each packet should not have memory conflicts or other hazards that stall the pipeline or cause packets to be dropped. Also, the latency of each task needs to be fixed with predetermined pipeline stages. Details of supported NN function kernels are as follows:

1) 2D Convolution: In 2D convolution, a small matrix of kernel weights *slides* over the 2D input data, performing an element-wise multiplication with the part of the input it is currently over, and then sums the results into a single output pixel. The weight kernel and 2D input data are preloaded into BRAM. The 2D input data is streamed into the kernel pipeline. In FCsN, halo exchange happens between the edges of the partitioned 2D input data and tasks carry corresponding data to be exchanged.

2) Dense and Normal Sparse Matrix Multiplication: With distributed matrix multiply between matrices A and B, in data-centric programming model matrix B can be treated as ''data" distributed in the system and matrix A decomposed into tasks. We recommend generating tasks with the sparse matrix if there

is one. The task carries operation information and the non-zero elements in the sparse matrix or a slot of dense matrix data.

3) Function Norms: Norms, such as layernorm and batch-norm [38], are realized through vector dot product (to calculate statistics such as sum and mean) and element-wise multiplication and addition (to scale and add biases). Their implementation can be modified from 2D Convolution and Matrix Multiplication kernels to achieve line rate.

4) Non-linear element-wise activation functions: Activation functions add non-linearity to the neural network models for improved prediction accuracy [39]. The activation function can be inserted in the computation pipeline acting on the output of the neuron.

D. Aggregation Function

Within GNN workloads, aggregation is an extreme case of Sparse Matrix Multiply with irregular data accesses for which it is hard to achieve streaming execution [40], [41]. Details of achieving streaming kernel execution at line rate with aggregation are described in this section.

1) Aggregation Function Operation: Aggregation is a matrix multiplication between an extremely sparse adjacency matrix ($>= 99\%$) and dense weight matrix. To facilitate streaming compute kernels with the data-centric model, the sparse adjacency matrix is decomposed into tasks carrying non-zero elements traversing in the network acting on the distributed weight matrix in the system. The equation *Output* = *AX* illustrates the operation of aggregation with the sparse matrix *A* and the dense matrix *X*. For each non-zero element in the task packet, one row of dense matrix *X* using col_id and one row of the output matrix using row_id is fetched.

2) Non-conflict Streaming Execution: Line-rate kernel execution requires a non-conflict, non-stall pipeline with several issues needing to be addressed. (1) Reading matrix *X* or output matrix may have conflicts between non-zero elements in each packet task. Each non-zero element needs to fetch data in one cycle to avoid pipeline stalls. However, there is no latency guarantee for memory accesses of multiple elements. (2) Writing back to the output matrix may result in conflicts between non-zero elements if more than one element is updating the same row. (3) Read after write (RAW) hazards may happen between reading and writing operation on the output matrix among different task packets within the pipeline.

To overcome these issues, the task spawner in the hardware runtime and BRAM optimizations in computing units cooperate. On-chip DDR access is slow and unpredictable, so the dense matrix data is loaded to BRAM. However, since each packet contains several non-zero elements, simply loading B and the output matrix to BRAM can not guarantee that several rows are fetched in one cycle. BRAM is therefore partitioned into interleaved blocks and the task spawner dynamically selects the corresponding element. Before generating tasks, the adjacency matrix is tiled into blocks as the storage format for sparse tensors where the data locality is improved. We are tiling the sparse matrix column-wise and with a block size of 20. This tiling helps the task spawner generate tasks with simple logic and BRAM access without conflicts (Figure 6).

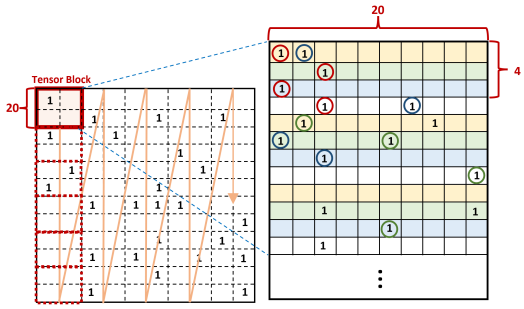

Fig. 6. Runtime Task Spawner

Accessing X Matrix: With a tensor block size of 20x20 (red block in Fig 6), the non-zero element in each column block only accesses the first 20 rows of the matrix *X*. We partition the BRAM cyclically so that every 20 rows of matrix *X* can be fetched in parallel. The task spawner generate tasks selecting non-zero elements within each block tile column-wise as indicated by the orange arrow in Fig 6.

Accessing and Updating Output Matrix: The non-zero selection needs to ensure non-conflicting *Output* access. The row id of the non-zero element determines which row of the output matrix will be read out and written to. As an example let each packet have 4 elements. We then partition the output matrix with a cyclic factor of 4 so every 4 rows of the output matrix can be independently read and written. The task spawner selects non-zero elements cyclic as factor of 4 to ensure interleaved data access on the output matrix BRAM. In Fig 6 the same color circle indicates non-zero elements in the same packet.

RAW hazard: To avoid RAW hazards, a RAW detector temporarily holds the last cycle's row id and vector value. If the non-zero element demands the row that is written back, it uses the stored vector in the RAW detector instead of fetching it from the *Output* BRAM.

V. Evaluation

A. Experiment setup

We evaluated the FCsN framework with NN kernel functions and NN applications using an Alveo U280 cluster with 2-4 nodes; systems with 8 and 16 nodes are evaluate with our cycle-accurate simulator with verified results collected in real systems. The FCsN hardware runtime and NN kernels are implemented in Vitis HLS. The baseline CPU results are evaluated with a 16-core 32-thread Intel® Xeon® Gold 6226R CPU. Current state-of-the-art distributed Pytorch-based NN applications use MPI as the backend. To eliminate the communication overhead and variance of different NIC or SmartNIC configurations, we implemented hand-tuned MPI code mapped to the cores of the same CPU. We also added a multi-node GPU with MPI using Nvidia Tesla V100s. The evaluations are with respect to four models: compute-centric (C-C) with MPI on CPU, GPU with MPI, computer-centric (C-C) with FCsN, and data-centric (D-C) with FCsN.

B. Performance and Resource Utilization

The baseline CPU MPI results evaluate the distributed Pytroch-based NN approach with the compute-centric model.

TABLE II
RESOURCE UTILIZATION, FREQUENCY: 294MHZ

	BRAM_18K	DSP	FF	LUT
NN Function Kernels	2,754 (66%)	2,724 (30%)	1,182,054 (42%)	834,876 (60%)
Hardware Runtime	717 (15%)	0 (0%)	87,813 (3%)	64,089 (3%)

We simplified and optimized the MPI code with the same computation and communication pattern as Pytorch. The multi GPU result is achieved by using MPI as the message passing interface. Compute-centric FCsN model follows the same computation and communication methods as the CPU baseline. However, the computation, control logic, and communication are offloaded onto the SmartNIC, with no CPU control involved. This result shows the improvement of the FCsN framework with detached host control. Lastly, FCsN with data-centric model indicates the performance of the data-centric model of streaming kernel execution with tightly fused computation and network pipeline detached from host control.

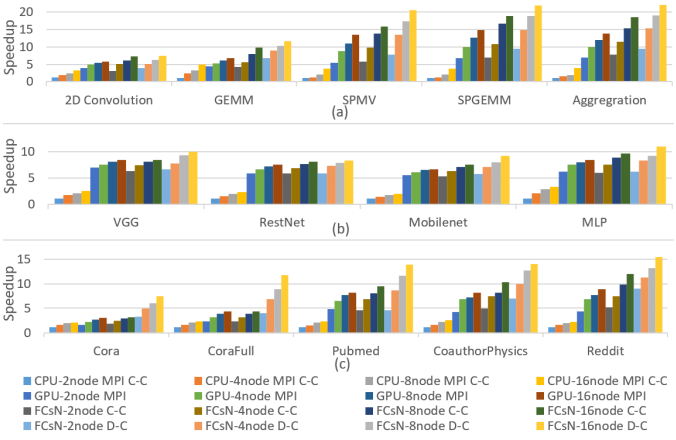

Fig. 7. NN Kernel and Application Model Evaluation Speedup

Figure 7 shows the normalized speedups for these evaluation models with NN kernels and NN applications. CPU with MPI and GPU with MPI have lower performance and scalability because of the communication and software overhead. The data-centric model gains more speedup than the compute-centric model as the system size scales up since data-movement and communication overhead increase. The FCsN data-centric model shows better speedups and scalability due to the streaming execution of task tokens that minimize data movement, and the avoidance of PCIe, software stack, host control, and synchronization.

1) Kernel Performance: Figure 7(a) shows the speedups of NN kernel functions with matrix size of 2048x2048. Since 2D convolution has less communication in 2D convolution and GEMM has regular data access and computation, FCsN with the data-centric model has limited speedup. However, in irregular kernels like SPMV, SPGEMM, aggregation, FCsN gains more speedup due to the offloaded control, asynchronous tasks, and streaming computation. In the 2D convolution function kernel, the speedup with FCsN in data-centric model is 2.3× compared with the CPU version, 1.27× compared with GPU and 1.1× compared with FCsN in compute-centric mode. The distributed GEMM kernel requires data movement between nodes. FCsN has less overhead handling communication than the CPU and GPU approaches. Compute-centric FCsN has a speedup of 2× compared to the CPU baseline. Data-centric FCsN provides higher performance than CPU, GPU, and FCsN in compute-centric with streaming computation with a speedup of 2.3× over the CPU baseline and 1.7× over the GPU. Sparse matrix-vector multiplication (SPMV), sparse matrix multiplication (SPGEMM) and aggregation have similar data distribution and execution. Computations are decomposed into tasks and vector or dense matrix are distributed as data in data-centric model. These kernels follow similar speedup trends over the baseline. The average speedup over CPU is 3.8× for compute-centric FCsN and 6.7× for data-centric FCsN.

2) Application Performance: Figure 7(b) shows the performance of neural network applications (VGG, RestNet, Mobilenet, MLP [42]–[45]) using FCsN provided function kernels. The speedups of VGG (2D-convolution), RestNet (2D-convolution), MobileNet (GEMM) and MLP (GEMM) are 3.36×, 3.55×, 4.3×, and 2.9×, respectively, for FCsN compute-centric over the CPU baseline. The speedup of FCsN data-centric over CPU baseline is 4×, 3.6×, 4.6× and 3.3×. We evaluated GNN models using the aggregation kernels with five datasets in Figure 7(c), Cora, CoraFull, Pubmed, CoautherPhysics, and Reddit [46], [47]. The size of datasets increases in order. The speedups of FCsN compute-centric over CPU are 1.52×, 1.86×, 4.82×, 5.08× and 8.5×. The speedup of FCsN data-centric over baseline are 5.38×, 5.08×, 6.04×, 6.64× and 10.13×.

3) Resource Utilization: Table II shows resource utilization of NN function kernels and the FCsN hardware runtime. The BRAMs and LUT resources are used largely to avoid memory access conflicts.

VI. CONCLUSION

We provide a user-friendly FPGA-Centric SmartNIC framework (FCsN) for Neural Networks, with a light-weight distributed hardware runtime and data-centric programming model that is completely detached from the CPUs. With the task circulation execution model, communication and computation are tightly fused and distribute kernel execution in a streaming manner at network line rate. A hardware-based FPGA-centric SmartNIC runtime enables asynchronous and fine-grained task scheduling which allows FCsN to be detached from host. FCsN leverages these characteristics to achieve high performance and efficiency for irregular and data-intensive neural network applications.

ACKNOWLEDGMENT

This work was supported by the Compute-Flow-Architecture (CFA) project under PNNL's Data-Model-Convergence (DMC) LDRD Initiative. The Pacific Northwest National Laboratory is operated by Battelle for the U.S. Department of Energy under Contract DE-AC05-76RL01830. This work was also supported, in part, by the NSF through award CCF-1919130.

References

[1] D. Sidler, G. Alonso, M. Blott, K. Karras, K. Vissers, and R. Carley, "Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*, 2015, pp. 36–43.

[2] P. Haghi, A. Guo, T. Geng, J. Broaddus, D. Schafer, A. Skjellum, and M. Herbordt, "A Reconfigurable Compute-in-the-Network FPGA Assistant for High-Level Collective Support with Distributed Matrix Multiply Case Study," in *2020 International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 159–164.

[3] ConnectX®-5 EN Card. [Online]. Available: https://support.mellanox.com/s/productdetails/a2v5000000052eSAAQ/connectx5-en-card

[4] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg, "Azure Accelerated Networking: SmartNICs in the Public Cloud," in *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'18. USA: USENIX Association, 2018, p. 51–64.

[5] Y. Zhu, Z. He, W. Jiang, K. Zeng, J. Zhou, and G. Alonso, "Distributed Recommendation Inference on FPGA Clusters," in *2021 31st International Conference on Field-Programmable Logic and Applications (FPL)*, 2021, pp. 279–285.

[6] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. P. Gopal, J. Gray, M. Haselman, S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Y. Xiao, and D. Burger, "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services," *IEEE Micro*, vol. 35, no. 3, pp. 10–22, 2015.

[7] H. Eran, L. Zeno, M. Tork, G. Malka, and M. Silberstein, "NICA: An Infrastructure for Inline Acceleration of Network Applications," in *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '19. USA: USENIX Association, 2019, p. 345–361.

[8] "Alveo U25 SmartNIC Accelerator Card," https://www.xilinx.com/products/boards-and-kits/alveo/u25.html.

[9] "Xilinx Alveo SN1000 SmartNIC," https://www.xilinx.com/applications/data-center/network-acceleration/alveo-sn1000.html.

[10] "Intel® Infrastructure Processing Unit (Intel® IPU) and SmartNICs," https://www.intel.com/content/www/us/en/products/network-io/smartnic.html.

[11] H. Shahzad, A. Sanaullah, and M. Herbordt, "Survey and Future Trends for FPGA Cloud Architectures," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, doi: 10.1109/HPEC49654.2021.9622807.

[12] C. Bobda, J. Mandebi, P. Chow, M. Ewais, N. Tarafdar, J. Vega, K. Eguro, D. Koch, S. Handagala, M. Leeser, M. Herbordt, H. Shahzad, P. Hofstee, B. Ringlein, J. Szefer, A. Sanaullah, and R. Tessier, "The Future of FPGA Acceleration in Datacenters and the Cloud," *ACM Transactions on Reconfigurable Technologies and Systems*, vol. 15, no. 3, pp. 1–42, 2022, doi: 10.1145/3506713.

[13] NVIDIA Mellanox BlueField SmartNIC for Ethernet. [Online]. Available: https://www.mellanox.com/files/doc-2020/pb-bluefield-smartnic.pdf

[14] Marvell® LiquidIO™ III. [Online]. Available: https://www.marvell.com/content/dam/marvell/en/public-collateral/embedded-processors/marvell-liquidio-III-solutions-brief.pdf

[15] Mellanox Innova-2 Flex Open Programmable SmartNIC. [Online]. Available: https://www.mellanox.com/files/doc-2020/pb-innova-2-flex.pdf

[16] Netronome® Agilio® SmartNICs. [Online]. Available: https://www.netronome.com/products/smartnic/overview/

[17] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13.

[18] D. Sidler, Z. István, and G. Alonso, "Low-latency TCP/IP stack for data center applications," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, 2016, pp. 1–4.

[19] D. Sidler, "In-Network Data Processing using FPGAs," PhD dissertation, Computer Science Department, ETH Zurich, 2019-09.

[20] D. Sidler, Z. Wang, M. Chiosa, A. Kulkarni, and G. Alonso, "StRoM: Smart Remote Memory," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3342195.3387519

[21] G. Sutter, M. Ruiz, S. López-Buedo, and G. Alonso, "FPGA-based TCP/IP Checksum Offloading Engine for 100 Gbps Networks," in *2018 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, Dec 2018.

[22] M. Ruiz, D. Sidler, G. Sutter, G. Alonso, and S. López-Buedo, "Limago: an FPGA-based Open-source 100 GbE TCP/IP Stack," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, Sep 2019, pp. 286–292.

[23] R. Jaganathan, K. Underwood, and R. Sass, "A configurable network protocol for cluster based communications using modular hardware primitives on an intelligent NIC," in *11th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2003. FCCM 2003.*, 2003, pp. 286–287.

[24] N. Zilberman, Y. Audzevich, G. Covington, and A. W. Moore, "NetFPGA SUME: Toward 100 Gbps as Research Commodity," *IEEE Micro*, vol. 34, no. 05, pp. 32–41, sep 2014.

[25] A. Caulfield, P. Costa, and M. Ghobadi, "Beyond SmartNICs: Towards a Fully Programmable Cloud: Invited Paper," in *2018 IEEE 19th International Conference on High Performance Switching and Routing (HPSR)*, 2018, pp. 1–6.

[26] K. D. Underwood, R. Sass, and W. B. Ligon, "Cost Effectiveness of an Adaptable Computing Cluster," *ACM/IEEE SC 2001 Conference (SC'01)*, pp. 30–30, 2001.

[27] V. Krishnan, O. Serres, and M. Blocksome, "COnfigurable Network Protocol Accelerator (COPA): An Integrated Networking/Accelerator Hardware/Software Framework," in *2020 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2020, pp. 17–24.

[28] W. Schonbein, R. E. Grant, M. G. F. Dosanjh, and D. Arnold, "INCA: In-Network Compute Assistance," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356153

[29] T. Hoefler, S. Di Girolamo, K. Taranov, R. E. Grant, and R. Brightwell, "SPIN: High-Performance Streaming Processing In the Network," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126970

[30] P. Haghi, T. Geng, A. Guo, T. Wang, and M. Herbordt, "FP-AMG: FPGA-Based Acceleration Framework for Algebraic Multigrid Solvers," in *28th IEEE International Symposium on Field-Programmable Custom Computing Machines*, 2020, doi: 10.1109/ FCCM48280.2020.00028".

[31] Z. He, D. Parravicini, L. Petrica, K. O'Brien, G. Alonso, and M. Blott, "ACCL: FPGA-Accelerated Collectives over 100 Gbps TCP-IP," in *2021 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC)*, 2021, pp. 33–43.

[32] P. Haghi, A. Guo, Q. Xiong, C. Yang, T. Geng, J. Broaddus, R. Marshall, D. Schafer, A. Skjellum, and M. Herbordt, "Reconfigurable switches for high performance and flexible MPI collectives," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 2, 2022, doi: 10.1002/cpe.6769.

[33] P. Haghi, A. Guo, T. Geng, A. Skjellum, and M. C. Herbordt, "Workload Imbalance in HPC Applications: Effect on Performance of In-Network Processing," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1–8.

[34] C. Tan, C. Xie, T. Geng, A. Marquez, A. Tumeo, K. Barker, and A. Li, "ARENA: Asynchronous Reconfigurable Accelerator Ring to Enable Data-Centric Parallel Computing," *IEEE Transactions on Parallel Distributed Systems*, vol. 32, no. 12, pp. 2880–2892, dec 2021.

[35] T. Geng, C. Wu, C. Tan, C. Xie, A. Guo, P. Haghi, S. Y. He, J. Li, M. Herbordt, and A. Li, "A Survey: Handling Irregularities in Neural Network Acceleration with FPGAs," in *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, 2021, pp. 1–8.

[36] Pynq. [Online]. Available: http://www.pynq.io/

[37] Xilinx Runtime Library (XRT). [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/xrt.html
[38] A. Rannen-Triki, M. Berman, V. Kolmogorov, and M. B. Blaschko, "Function Norms for Neural Networks," in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 748–752.
[39] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," 2018. [Online]. Available: https://arxiv.org/abs/1811.03378
[40] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "AWB-GCN: A Graph Convolutional Network Accelerator with Runtime Workload Rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 922–936.
[41] T. Geng, C. Wu, Y. Zhang, C. Tan, C. Xie, H. You, M. Herbordt, Y. Lin, and A. Li, "I-GCN: A Graph Convolutional Network Accelerator with Runtime Locality Enhancement through Islandization," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1051–1063. [Online]. Available: https://doi.org/10.1145/3466752.3480113
[42] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *CoRR*, vol. abs/1409.1556, 2015.
[43] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
[44] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *CoRR*, vol. abs/1704.04861, 2017.
[45] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis, "Multilayer Perceptron and Neural Networks," vol. 8, no. 7, p. 579–588, jul 2009.
[46] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective Classification in Network Data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2157
[47] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *CoRR*, vol. abs/2003.00982, 2020. [Online]. Available: https://arxiv.org/abs/2003.00982