Exploring HW/SW Co-Design for Video Analysis on CPU-FPGA Heterogeneous Systems

Xiaofan Zhang, *Student Member, IEEE,* Yuan Ma, *Student Member, IEEE,* Jinjun Xiong, *Senior Member, IEEE,* Wen-mei Hwu, *Fellow, IEEE,* Volodymyr Kindratenko, *Senior Member, IEEE,* and Deming Chen, *Fellow, IEEE*

***Abstract*—Deep neural network (DNN) based video analysis has become one of the most essential and challenging tasks to capture implicit information from video streams. Although DNNs significantly improve the analysis quality, they introduce intensive compute and memory demands and require dedicated hardware for efficient processing. The customized heterogeneous system is one of the promising solutions with general-purpose processors (CPUs) and specialized processors (DNN Accelerators). Among various heterogeneous systems, the combination of CPU and FPGA has been intensively studied for DNN inference with improved latency and energy consumption compared to CPU + GPU schemes and with increased flexibility and reduced time-to-market cost compared to CPU + ASIC designs. However, deploying DNN-based video analysis on CPU + FPGA systems still presents challenges from the tedious RTL programming, the intricate design verification, and the time-consuming design space exploration. To address these challenges, we present a novel framework, called EcoSys, to explore co-design and optimization opportunities on CPU-FPGA heterogeneous systems for accelerating video analysis. Novel technologies include 1) a coherent memory space shared by the host and the customized accelerator to enable efficient task partitioning and online DNN model refinement with reduced data transfer latency; 2) an end-to-end design flow that supports high-level design abstraction and allows rapid development of customized hardware accelerators from Python-based DNN descriptions. 3) a design space exploration (DSE) engine that determines the design space and explores the optimized solutions by considering the targeted heterogeneous system and user-specific constraints; and 4) a complete set of co-optimization solutions, including a layer-based pipeline, a feature map partition scheme, and an efficient memory hierarchical design for the accelerator and multi-threading programming for the CPU. In this paper, we demonstrate our design framework to accelerate the long-term recurrent convolution network (LRCN), which analyzes the input video and output one semantic caption for each frame. EcoSys can deliver 314.7 and 58.1 frames per second (FPS) by targeting the LRCN model with AlexNet and VGG-16 backbone, respectively. Compared to the multithreaded CPU and pure FPGA design, EcoSys achieves 20.6× and 5.3× higher throughput performance.**

***Index Terms*—Heterogeneous System, HW/SW Co-Design, DNN, CAPI, Acceleration.**

I. Introduction

VIDEO content analysis is one of the most challenging applications that allows computers to understand the human world, which contains copious incomplete and non-structural information. By adopting DNNs, such as convolutional neural networks (CNN) and recurrent neural networks (RNNs), the quality of video content analysis has been greatly improved. Recently, we have seen a rapid development of DNNs for image/video recognition tasks, and among them, the long-term recurrent convolutional network (LRCN) is one of the most prominent solutions for video content analysis, such as performing activity recognition and content captioning for the input videos [1]. Such a capability allows machines to handle more real-life applications that originally require human efforts. For example, the videos captured by the surveillance systems can be fast examined by machines to identify particular dangerous scenes without manual interventions. Some emerging applications, such as video and image annotations, can also benefit from the development of DNN-based video analysis in order to generate the desired output descriptions automatically.

The workflow of LRCN is shown in Fig. 1. The input video frames are first passed to a CNN to extract its spatial features. These features are fed into an RNN composed of multiple long short-term memory (LSTM) layers to finally produce a video content description. The advantages of LRCN come from its hybrid neural network structure combining both CNN and RNN layers: the CNN is first used to capture the input's spatial information, while the following RNN takes these spatial features sequentially and generates text descriptions. Although LRCN is a powerful tool for video analysis, it involves more complex network structures and requires more intensive compute and memory demands during inference compared to a single CNN or RNN. To efficiently handle such a unique DNN, customized heterogeneous systems are developed with the hardware combination of both CPUs and dedicated DNN accelerators. A suitable hardware platform is important for running LRCN efficiently. As a reconfigurable platform, FPGA becomes a satisfactory hardware candidate to host the LRCN than the conventional computer system due to its fine-grained parallelism and the low-power budget.

A typical implementation of the LRCN includes an AlexNet [2] for the CNN and multiple LSTM layers [3] for the RNN. It contains 2.22 billion floating-point operations and 86.56 million synapse weights for processing one input video frame. LRCN can be extended to use more advanced DNN backbones by modifying the CNN part, such as using VGG [4] and ResNet [5]. Such a unique DNN combination exhibits

Xiaofan Zhang, Yuan Ma, and Deming Chen are with the Department of Electrical and Computer Engineering, University of Illinois Urbana-Champaign, IL, 61801. E-mail: {xiaofan3, yuanm2, dchen}@illinois.edu

Jinjun Xiong is with IBM T.J. Watson Research Center, NY, 10598. E-mail: jinjun@us.ibm.com

Volodymyr Kindratenko is with National Center for Supercomputing Applications (NCSA), University of Illinois Urbana-Champaign, IL, 61801. E-mail: kindrtnk@illinois.edu

Wen-mei Hwu is with NVIDIA, CA, 95051. E-mail: whwu@nvidia.com

Manuscript received December XX, 2020; revised XXX XX, XXXX.

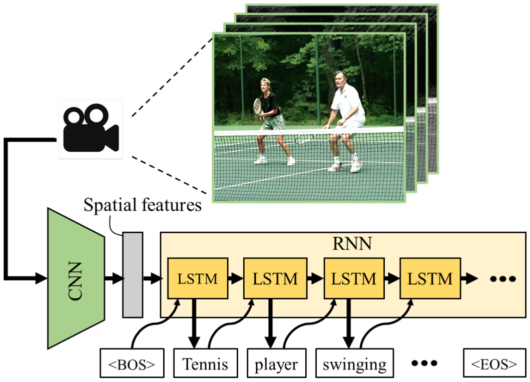


Fig. 1. LRCN adopts both CNN and RNN for video content analysis. The CNN generates spatial features of the inputs while the RNN outputs a sentence as the video content description in a recurrent manner, which starts with the entry <BOS> (beginning of sequence) and ends up with <EOS> (end of sequence).

significantly different layer characteristics with respect to the computational complexity (the number of operations required in one layer) and memory demands (the amount of data fetched and stored in a certain period). For example, the convolutional (CONV) layers are computationally intensive, so accelerating these layers is often limited by the available compute units. In contrast, the fully-connected (FC) layers and LSTM layers are highly memory-intensive, where on-chip memory size and the external memory access bandwidth dominate their achievable performance. With such characteristics, using general-purpose processors may fail to deliver satisfactory performance and efficiency for video content analysis using LRCN. A customized heterogeneous system with dedicated DNN hardware accelerators becomes essential to provide optimized solutions.

In addition to CPUs, heterogeneous systems developed recently also include various hardware accelerators, such as Graphics Processing Units (GPUs), Tensor Processing Units (TPUs), and Neural Network Processing Units (NPUs), to attract and leverage numerous demanding applications. Among these accelerators, the customized accelerator, such as the FPGA- or ASIC-based design, is one of the most promising candidates for running DNN inference. They can be fully customized to implement the neural network functionality with improved performance or efficiency compared to CPU- and GPU-based designs. For example, authors in [6] compare an FPGA-based DNN accelerator to CPU, GPU, and embedded GPU designs. By accelerating the same VGG16 network, the FPGA design is 1.4× and 2.0× faster than the solutions using CPU and embedded GPU. Although the GPU-based design achieves 13.0× higher performance, it requires 26.0× more power consumption. Similarly, customized FPGA-based DNN accelerators generated by DNNBuilder are more efficient and with better real-time performance compared to the GPU solutions [7]. When benchmarking using the same DNN, authors in [7] indicate that the accelerator implemented on an embedded FPGA delivers 2.0× higher power efficiency than an embedded GPU design, while they can reach more than 4.3× higher efficiency using a mid-range FPGA comparing to a desktop GPU. By reaching the same level of throughput, the FPGA-based designs are likely to have lower batch size requirements, which exhibit better real-time performance. Once design flexibility, time-to-market, and non-recurring engineering costs are considered, FPGA presents even higher potential and becomes a more suitable choice than ASIC [6]–[9]. However, using FPGA for domain-specific accelerator designs generally requires RTL programming, hardware verification, and precise resource allocation, all of which can be time-consuming and challenging even for seasonal FPGA developers. These negative aspects often hinder FPGA's adoption by application developers who are used to working on high-level programming and have less experience deploying targeted applications onto a system containing FPGAs.

One of the popular directions to alleviate such difficulties is to adopt a higher level of design abstraction for FPGA development, which requires fewer lines of code and gives faster simulation results. For example, High-level Synthesis (HLS) allows the use of behavioral-level programming languages (C/C++) to be the abstract descriptions of hardware functions. It thus helps reduce considerable amounts of lines of code compared to RTL programming languages (7× for a hardware design with one million logic gate [10]) and significantly improves the design efficiency [11]. With the HLS design flow, customized DNN accelerators have been developed to meet the needs of various AI applications, such as accelerating image classification [12]–[14], object detection [15]–[17], and language translation [18]–[20]. As most DNNs are developed by machine learning frameworks using Python, it allows an even higher design abstraction level and creates a greater gap between DNN designs on software and their hardware deployments. To bridge the gap, researchers have been investigating end-to-end design flows that take DNN definition files (compatible with the machine learning frameworks) as inputs and automatically generate the hardware accelerator designs to improve accelerator design efficiency [7], [21].

Other challenges of implementing a video analysis system come from the massive amount of memory consumption for caching the DNN parameters (weights and biases), intermediate results (feature maps), and inputs (video frames). Storing all the data using on-chip memory seems to be taken for granted. However, the required memory space of LRCN using AlexNet can reach 346 MB, which significantly exceeds the FPGA's on-chip memory capacity. Therefore, most of the parameters and intermediate results have to be placed off-chip and require sophisticated memory management to diminish extra data transfer overheads. Challenges also come from real-life video analysis applications, especially when the DNN parameters must be updated during continuous learning to keep adopting the real scenes. The goal is to absorb a new dataset and continually refine the DNN parameters in real-time. Such a specific requirement causes extra data transfers between the main memory on the host side and the external accelerator memory on the accelerator side. Besides, maintaining the data coherence can become problematic, especially when the tasks are distributed to different hardware devices. In this case, the system is required to guarantee the most up-to-date parameters are coherent and can be accessed timely by all devices. Therefore, the more frequently the parameters get updated, the worse performance could achieve.

To address the above-mentioned challenges, we propose EcoSys, a novel framework to develop customized heterogeneous systems containing both CPU and FPGA. EcoSys proposes an end-to-end design flow to connect the Python-based high-level DNN descriptions with their board-level FPGA implementations. It also integrates a comprehensive design space exploration (DSE) to generate suitable task partition schemes and hardware configurations for achieving optimized performance given targeted CPU + FPGA systems. It can also create a coherent memory space for the host CPU and the customized accelerator and eliminate data transfer latency between these devices. Detailed contributions of this paper are further discussed in Section III.

II. Related Work

LRCN is a hybrid neural network model using both CNN and RNN that can together learn both the space information in vision and the sequential information in time. As depicted in Fig. 1, video frames are entered sequentially into the system and first processed by a CNN for the extraction of visual features. These features of the incoming image frame are passed to the RNN module to generate proper descriptions. The original combination published in [1] adopts AlexNet for the CNN and LSTM layers for the RNN, but it can be updated to use more update-to-date CNNs (e.g., VGG, ResNet, etc.) for better feature extraction. The unique network structure provided by LRCN can be widely applied to handle video analysis tasks that require capturing features in both spatial and temporal patterns.

To leverage the complicated DNN workloads, we have seen extensive studies in the DNN hardware accelerator. Among these designs, FPGA-based solutions have been rapidly developed and become one of the most promising solutions to improve performance and efficiency [7], [15], [22], [23]. Designs presented in [24] explore the design space of loop optimizations in a CNN implementation to locate the best implementation point. To improve the hardware efficiency, the authors in [6], [25] investigated dynamic quantization schemes for quantizing both DNN parameters and intermediate feature maps. The designs in [26], [27] also support binary and ternary quantization, which further relax the intensive computational pressure by replacing the hardware-intensive floating-point multiplications with logical operations. Other optimizations include implementing fast CONV algorithms on FPGA, such as using Winograd-based solutions and fast Fourier transform (FFT) to replace the original spatial CONV operations [13], [28], [29]. Recently, we have seen more and more designs that adopt both hardware and software optimizations. In [9], the targeted DNN is first compressed and then deployed onto FPGA, while authors in [15], [16] propose a hardware-software co-design strategy to alleviate design contradictions coming from the resource-demanding DNN applications and resource-constrained hardware.

Researchers also focus on building DNN accelerator design frameworks for improved hardware design efficiency when deploying DNN workloads on FPGAs [7], [30]–[32]. Most of the frameworks contain automation flows that support a high abstraction level of design entries and deliver hardware implementations on FPGAs by assembling RTL or HLS building blocks. Following this strategy, the design in [21] proposes a unified representation for CONV and FC layers to facilitate accelerator modeling for DNNs developed by Caffe. To improve accelerator design and optimization, DNNBuilder is proposed to provide an end-to-end automation tool for building and prototyping high-quality FPGA-based accelerators [7]. It allows auto-optimization of the customized accelerators to deliver real-time DNN inference even for resource-constraint hardware. A framework proposed in [30] incorporates systolic arrays for improved computing efficiency, while the design in [32] proposes a new architecture template by combining both pipeline and recurrent architecture. Researchers also target various heterogeneous systems and propose design frameworks to provide systematic design and optimization solutions. In [33], authors introduce an OpenCL-based framework to support a heterogeneous cluster with CPUs and GPUs for accelerating object detection. In [34], a schedule exploration and optimization framework is published for tensor computation on heterogeneous systems. In addition, authors in [16] publish a design framework targeting embedded systems with CPU + FPGA and CPU + GPU to leverage popular real-life AI applications including object detection and object tracking.

Instead of solely optimizing the DNN hardware implementation, researchers also explore the co-optimization opportunities of both hardware and software. For example, authors in [15] introduce a uniform intermediate representation for DNN and accelerator co-design, while authors in [16] develop a hardware-efficient DNN model and high-performance customized accelerator following the hardware/software co-design strategy to satisfy predefined performance targets and hardware constraints. To improve the co-design efficiency, there have been growing interests in using neural architecture search (NAS) with hardware performance feedback to generate hardware-efficient DNNs and higher-level design abstraction to develop customized accelerators [35]–[37]. For example, a fast and a slow search are iteratively adopted in [36] to speed up NAS and maintain model accuracy. In [37], gradient descent is applied to enable a fast search for suitable DNN structures and corresponding hardware accelerators.

III. Contributions

In this paper, we present EcoSys, a novel framework to deliver customized heterogeneous system design for DNN-based video analysis with improved performance and efficiency. This work is partially extended from our previous work published in FPL'17 [25] where DNN accelerator is solely implemented on FPGA. Compared to our previous design, EcoSys targets a heterogeneous system design with both CPU and FPGA to better leverage network models with both CNNs and RNNs. With the built-in DSE engine, EcoSys is able to perform system-scale task partitioning given available hardware resources and targeted DNNs and make full use of different devices. We then improve the accelerator architecture by adopting a layer-based pipeline and feature map partition scheme to achieve better results than our previous design. By supporting Python-based DNN descriptions as inputs, EcoSys substantially raises the design abstraction level for building customized accelerators and provides an efficient approach to design, optimize, and integrate an FPGA-based DNN accelerator into a heterogeneous system. With the proposed framework, we can significantly speed up the design process and improve the performance of CPU + FPGA systems. To summarize, the main contributions of this work are as follows:

1) EcoSys is the first hardware-software co-optimization framework supporting CAPI-based coherent memory space [38] and an end-to-end design flow for building customized heterogeneous systems targeting DNN-based video analysis applications. It delivers a more efficient approach to develop customized accelerators from a higher level of design abstraction and an easier way to support data sharing without worrying about the memory coherence issues and excess data transfer latency.
2) We create a coherent memory space shared by the host CPU and the FPGA accelerator. It intends to provide an ample memory space to accommodate complicated DNN models and reduce data transfer latency and synchronization from one device to the other. In our design, the customized accelerator is allowed to directly access the same memory space shared by the host CPU, which helps reduce up to 19.96% of latency compared to the CPU + FPGA system with a traditional connection.
3) We present an end-to-end design flow from Python-based DNN designs in the deep learning framework to their board-level implementation on the targeted FPGA. It includes a highly configurable accelerator architecture to leverage DNN implementations and essential supporting modules, such as DNN layers and CAPI compatible memory interface, to ensure efficient acceleration of the targeted workloads. The proposed flow enables designing and optimizing the whole system on a high level of abstraction with improved developing efficiency.
4) We introduce a DSE engine to leverage efficient explorations within the predefined space and deliver the optimized accelerators by considering various constraints, such as available resources and the targeted workload. The proposed DSE engine also helps deliver task partition schemes, where targeted workloads can be distributed to different hardware devices in the heterogeneous system.
5) We leverage a complete set of co-optimization methods to push the achievable performance further. For accelerator design, we include a layer-based pipeline to boost throughput performance, a feature map partition scheme to minimize on-chip memory utilization, highly configurable IPs for layer dedicated hardware designs, and an efficient memory hierarchical design for timely data supply. Regarding the CPU side, we introduce OpenMP templates for multithreading layer implementation.

IV. Design Challenges of Accelerating Video Analysis

The rapidly developing DNNs keep improving the output quality of video analysis algorithms. They also deliver more complicated algorithms with enormous compute and memory demands, which require dedicated hardware accelerators to enable real-life deployments. Before introducing the detailed approaches for building the accelerators, in this section, we summarize three design challenges from DNN structure, real-life requirements, and hardware implementation.

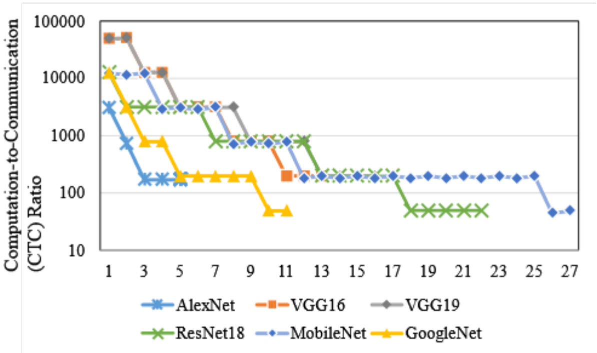


Fig. 2. CTC variations of network layers in six popular DNN models. The layer closes to the input is numbered as 1. Noted that we consider each inception module in the GoogleNet as one "layer" and omit the fully-connected layers (with CTC = 1) from all DNNs for clearer presentation.

TABLE I
Percentage of computation and memory consumption in LRCN

Layer	Compute resource	Memory resource
CONV	60.06%	2.69%
FC	5.29%	67.73%
LSTM	34.65%	29.58%

A. Diverse DNN layers

The unique combination of CNN and RNN creates unbalanced demands for different types of layers in LRCN. These layers exhibit different characteristics in consuming compute and memory resources. Regarding an LRCN model with Alexnet backbone and 15 iterations of LSTM layers, processing one video frame costs 2.22 Giga operations (GOP) and consumes 86 million DNN parameters occupying 346 MB of memory. As shown in Table I, we break down the resource demands for three major types of layers as CONV and FC layers in the CNN and LSTM layers in the RNN. The computation demand is dominated by the CONV layers with 60.06% of the total computation, while the memory consumption mostly comes from the FC layers, peaking at 67.73% of overall required parameters.

DNN layers are considerably different regarding their compute and memory-access patterns, even for layers in the same model. We investigate six popular DNNs and summarize the computation-to-communication (CTC) ratio (a metric for indicating arithmetic intensity) of each layer with parameters in Fig. 2. We define the Layer's CTC ratio as the number of multiply-accumulate operations supported by one parameter fetched from memory. Layers with higher CTC ratios mean that they are expected to be compute-bounded, while layers with lower ratios indicate that they are more likely memory-bounded. We observe a CTC downward trend for each model

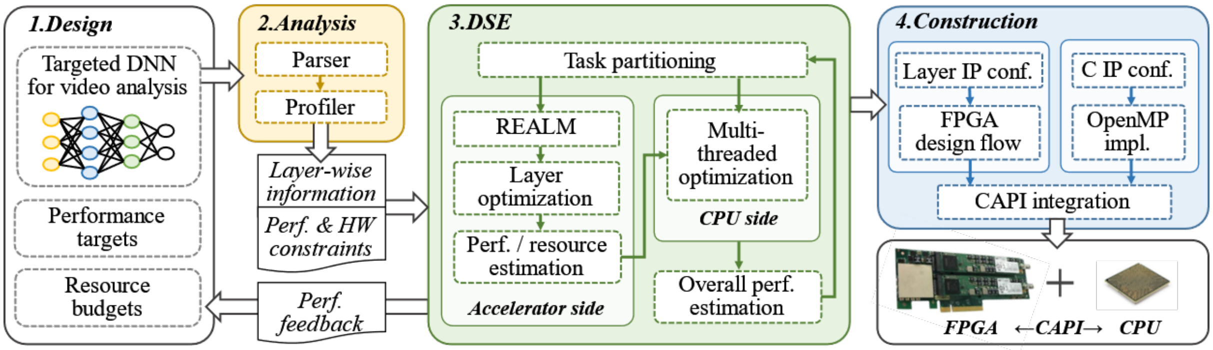


Fig. 3. The proposed EcoSys design flow with *Design*, *Analysis*, *DSE*, and *Construction* steps to deliver optimized CPU + FPGA heterogeneous systems for accelerating video analysis.

starting from the front part of DNN (close to the input layer). The gaps between the maximum and the minimum CTC are significant for all DNN models. For example, the maximum CTC is 255.7× and 254.7× higher than the minimum one in VGG16 and ResNet18, respectively. Such diverse layer characteristics pose tough challenges for efficient hardware accelerator design and require the proposed design to have great adaptability for different layers with expected performance.

B. Real-life application requirements

By targeting video analysis, accelerators are required to handle streaming inputs and deliver satisfactory throughput to match the video frame rate, such as 30 or 60 frames per second (FPS). This type of application also needs to support frequent user interactions, making real-time response indispensable. The design difficulty then lies in where the proposed hardware accelerator needs to deliver high throughput without using large batch sizes since the extra delay in collecting batch inputs may fail to meet the real-time requirement.

Another challenge comes from the continuous learning demands in real-time. Like most neural network models specialized in the computer-vision domain, LRCN is first trained on a benchmark dataset and then deployed on hardware with the fixed, static parameters. To better adapt actual application scenarios, the DNN model needs to be continuously updated by using additional real-life data. In this case, DNN parameters can not be treated as read-only data, and the proposed hardware accelerator is required to provide an efficient approach for parameter update.

C. Hardware implementation difficulties

Major layers, which dominate the compute and memory resources, such as CONV, FC, and LSTM layers, are constructed by multiple nested loops from the algorithmic perspective. They can be optimized during implementation on hardware, such as using loop unrolling to increase the parallelism and loop pipelining to improve the throughput performance. Although developing accelerators on a behavior level can alleviate the design effort, actual hardware designs may not always follow the designers' intentions. For example, the loop unrolling directive in HLS can be invalid if complex data dependencies are found in that loop. Besides, an optimal hardware implementation always requires careful resource allocation since the DNN layers vary significantly in compute and memory demands. If a DNN layer is memory-bounded, its achievable performance is less affected by the allocated computation resources but by how frequently it accesses the memory and whether enough memory access bandwidth can be provided. Thus, designers need to understand DNN layer structures and customized hardware accelerator design and implementation, along with a decent resource allocation scheme to deliver an optimized hardware design.

V. The proposed EcoSys framework

To overcome these challenges, we propose a novel design framework called EcoSys to explore the customized heterogeneous system designs for running video analysis with optimized performance. The overall design flow is shown in Fig. 3, which includes four steps to deliver the hardware implementation for the targeted DNN.

In the *Design* step, a targeted video analysis network is developed using deep learning frameworks. After training on the targeted dataset with satisfactory inference accuracy, the DNN model is ready for hardware implementation. Its Python-based definition files and parameters are passed to the next step for detailed analysis. Meanwhile, we can set up a performance target (e.g., throughput performance) and provide hardware specifications (e.g., available resources of the targeted CPU + FPGA system).

In the *Analysis* step, the proposed framework starts model analysis to collect layer-wise information. The built-in parser is responsible for understanding the input DNN architecture, including the network's input/output shape, layer type, layer configuration, etc. The profiling function then captures layer's compute and memory access patterns, such as the number of operations and CTC ratio, to outline the input model. The *Analysis* step also collects the targeted hardware specification to clarify resource constraints and the desired performance target.

When all the information is ready, the *DSE* step is launched to explore optimized configurations for accelerating targeted DNNs on our customized heterogeneous system. Since our proposed system includes an FPGA-based accelerator and a host CPU, the first task is to partition the input DNN and distribute workloads to different devices. On the accelerator side, resource allocation is executed to provide hardware configuration guidelines given the design space and available resources, such as DSPs (compute resources), BRAMs (on-chip memory resources), and external memory access bandwidth. Following these guidelines, layer optimization is performed to select the most suitable configuration for every pre-defined configurable layer IP within the resource constraints. Performance estimation of the accelerator is then provided to the CPU optimizer as performance calibration. On the CPU side, the optimization goal is to match the accelerator's performance since the optimized system-level performance requires perfect coordination among all different devices. Overall performance estimation is then provided as feedback to adjust the task partitioning. Eventually, the *DSE* step generates the optimized solution to configure the customized system given targeted workloads and available hardware resources. If the result is still not satisfactory, feedback with performance estimation will be sent to the *Design* step for model or hardware alternations. With such a feedback mechanism, we can significantly reduce the development time because we do not need to wait to complete hardware implementation before changing the design.

Design guidelines are then passed to the *Construction* step. Regarding the accelerator design, layer IP configuration is responsible for building DNN accelerators with pre-defined IP templates written by Verilog hardware description language. These IPs are highly configurable to guarantee adaptability and scalability when targeting different DNNs with various FPGA resource budgets. After going through the FPGA design flow, we generate the hardware instance of the customized accelerator. Meanwhile, on the CPU side, C-level design templates (DNN layer IPs described in C programming language) are configured according to the optimization guidelines from the *DSE*. We then use an Open Multi-Processing (OpenMP) API to execute these templates by taking advantage of the multi-core CPU architecture. Eventually, the accelerator and the host CPU are integrated with CAPI and form a customized heterogeneous system.

VI. Customized Accelerator Architecture

A. Architecture overview

As shown in Fig. 4 (a), the targeted DNN workloads are first partitioned into two parts: the first part starting with the CNN mapping to the accelerator and the second part mapping to the CPU. The task partitioning scheme is determined by the DSE (step 3 of the proposed flow) by considering various factors, such as compute and memory demands of the input DNN and available resource budgets. Fig. 4 (a) only presents one of the possible schemes where the split point is exactly between CNN and LSTM. It is possible to partition within the CNN where the last few CNN layers will be mapped to CPU. We limit the maximum split parts to two and always map the first part to FPGA and the second part to CPU. Such a design may restrict system-level variants, but it helps deliver a concrete CPU + FPGA architecture template that allows more specific optimization strategies to be applied. More importantly, it is suitable for our targeted CNN + RNN workloads. The first part is more compute-intensive (with higher CTC ratios according to Fig. 2) which the customized hardware accelerator can better address.

We adopt a CAPI-based coherent interconnection between the host CPU and the accelerator. These hardware devices are connected with coherent memory space for improved memory capacity and lower data transfer latency. In our design, accelerators are granted privileges to access the memory banks with the same effective addresses from the host CPU without involving direct communications. This feature effectively overcomes the limitations caused by low memory capacity and data synchronization procedures. It also reduces the number of data transfers because the enormous video inputs no longer need to be transferred to the accelerator's own memory space at run-time.

Since the accelerator and CPU can work seamlessly, customized system designs generated by EcoSys can achieve a layer-based pipeline structure for improved throughput performance as shown in Fig. 4 (b). Major layers in the CNN dominating the computation or memory consumption (e.g., CONV, Pooling, FC layers) are constructed as individual pipeline stages while the lightweight layers (e.g., activation layers and batch normalization layers) are aggregated to their neighboring major layers. We also adopt a fine-grained pipeline design from [7] to increase the overlap between pipeline stages, which equivalently reduces the initial latency. On the other side, layers in the RNN are treated as one pipeline stage because of their recurrent behaviors. If this part is mapped to CPU as the example showing in Fig. 4 (b), it will be launched after results from the previous layer are available in the coherent memory space.

B. Customized accelerator design

For the proposed accelerator design, EcoSys generates architecture units (AU) as pipeline stages corresponding to major DNN layers in the targeted network. In each AU, EcoSys instantiates three major hardware components, including a compute engine (CE), a weight buffer (WBuf), and an input buffer (InBuf). As shown in Fig. 4 (c), the yellow area in the accelerator architecture represents the proposed CE, which is highly configurable and responsible for the DNN layer operations. It supports a two-dimensional parallelism scheme to perform parallel processing for each layer following the input and output parallelism factor called *cpf* and *kpf*. It means *cpf* number of input channels and *kpf* number of output channels are processed simultaneously in one CE with the total parallelism factor as *cpf* × *kpf*. Inside each CE, there are *kpf* process elements (PEs) instantiated to handle computations. We will cover more details of the CE design in subsection VI-B1. To ensure sufficient data supply to the CE unit, we utilize input and weight buffers for on-chip data buffering. The input feature maps from the system input or previous AU are passed horizontally from the left and kept in the InBuf. To avoid buffering an entire feature map within each AU, we adopt a partition scheme published in [7] to keep a subset (several slices on the height-depth plane) of the

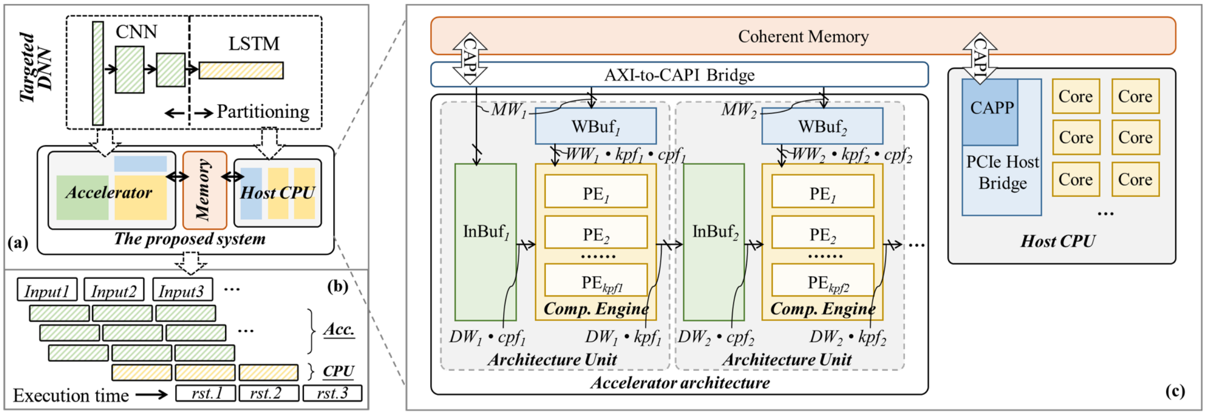


Fig. 4. The proposed customized heterogeneous system for accelerating DNN workloads specialized in video analysis.

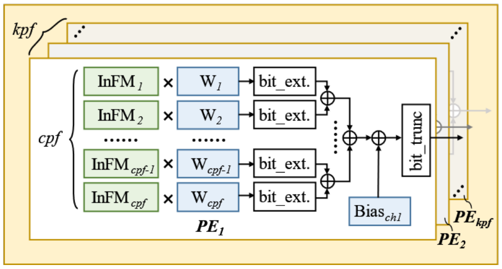


Fig. 5. Highly configurable compute engine (CE) design for diverse DNN compute demands.

input feature map on-chip. We will introduce this method in subsection VI-B2. Meanwhile, a fraction of DNN parameters corresponding to the current computation is transferred from the external coherent memory through an AXI-to-CAPI bridge and stored in the WBuf. The bitwidth for memory access (MW) is also configurable to satisfy various bandwidth demands from different AU designs. More details regarding memory management will be presented in subsection VI-B3. The proposed design also supports dedicated data quantization schemes for each AU with different bit-width combinations for input data (DW) and parameter (WW). EcoSys can generate accelerators with great adaptability and scalability to overcome challenges when handling the demanding DNN-based video analysis workloads with these configurable parameters.

1) Computation engine (CE) design: CE is the critical component for handling computations in every AU. Shown in Fig. 5 is the detailed structure in the proposed CE, which provides two-dimension parallel processing along with input and output channels of the targeted DNN layer. Assuming a CE with parallelism configured as *cpf* and *kpf*, *cpf* pairs of input and DNN weight are passed from the on-chip buffers and handled by the first process element (PE_1) (which is a sub-module in every CE) for multiply-accumulate (MAC) operations, and the generated results are produced along the first output channel dimension. Meanwhile, $PE_2 \sim PE_{kpf}$ are working on the same inputs but different DNN parameters to generate results for the next *kpf*-1 output channels. In this case, *kpf* PEs are instantiated providing a total parallelism factor as *cpf* $\times$ *kpf*. To support a dedicated quantization scheme for each pipeline stage, the proposed CE also contains a bit-extension and bit-truncation unit. With the configurable IP design, EcoSys can instantiate dedicated CEs to handle various layers operations in fulfilling the desired performance within the FPGA resource constraints.

2) Feature map partition: When handling DNN-based video analysis, the input feature can be enormous, especially when targeting video frames with large resolution. EcoSys fellows a feature map partition strategy called column-based cache [7] to effectively utilize the scarce on-chip memory by keeping a subset of feature maps instead of the entire ones. Assuming a 3-dimensional input feature map ($D \times H \times W$ for depth, height, and width) in a CONV layer with kernel size $D \times K \times K$ and stride S, the column-based cache only needs to keep $K + S$ slices of the input feature map to start sliding window operations along the H dimension. Each slice is a $D \times H$ plane and the input elements cached on-chip is $D \times H \times (K + S)$. After computation with the cached feature map subsets, the input buffer will release the oldest slice and start buffering the new one and supporting CE's continuous operations.

3) Memory hierarchy design: To improve the memory access efficiency, the data access patterns need to be simple and straightforward, which means data can be fetched following consecutive locations in memory. To ensure this, the multi-dimensional DNN parameters are re-ordered into a linear sequence that follows this order of computation. This preprocessing can guarantee that data locality is exploited when the CE instance accesses weight data and thus improve the throughput of access.

To further hide the effects of off-chip memory access latency, we overlap the computation with communication. When a CE instance is consuming weights already fetched, fetch requests for the new weights are sent out concurrently. As shown in Fig. 6, we design a hierarchical memory system to meet this requirement by instantiating FIFOs in the path connecting the on-chip buffers to external memory. The FIFOs pre-fetch a few parameters first, and for every FIFO row of parameters consumed by CE, a new row is fetched at the back

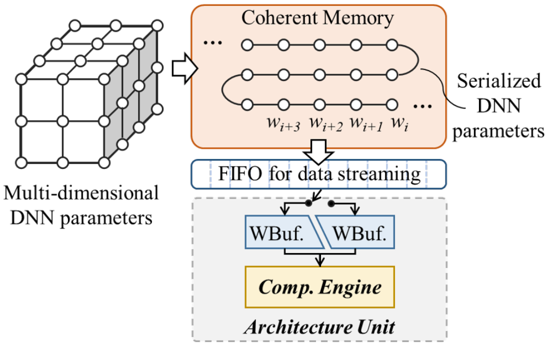

Coherent Memory
…
…
w_{i+3} w_{i+2} w_{i+1} w_i
Serialized DNN parameters
Multi-dimensional DNN parameters
FIFO for data streaming
WBuf.
WBuf.
Comp. Engine
Architecture Unit

Fig. 6. Memory hierarchy design for timely data delivery

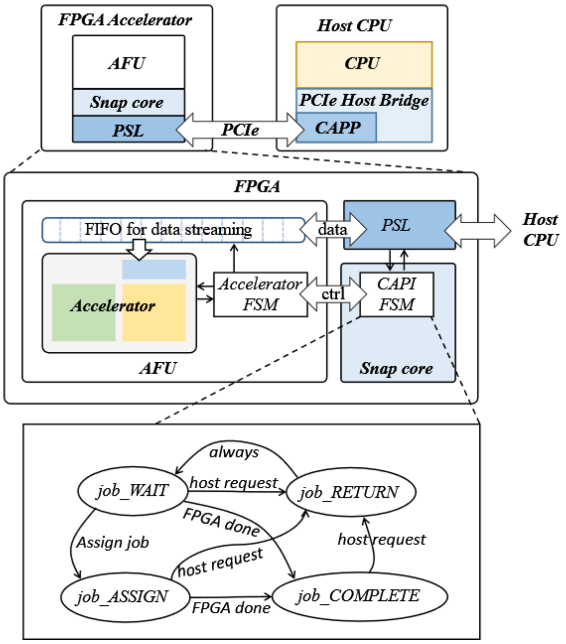


Fig. 7. Block diagrams for CAPI-compatible FPGA-based accelerator design.

of the FIFO. In addition, the weight buffers are instantiated as ping-pong buffers to hide a few cycles of access time between the proposed CE and the FIFO.

C. CAPI integration

We use Coherent Accelerator Processor Interface (CAPI) [38] for connecting the customized FPGA-based accelerator and the host CPU. CAPI divides the load of the traditional driver into two proxies as one on the host side, named Coherent Attached Processor Proxy (CAPP), while the other on the FPGA accelerator side, called Power Service Layer (PSL). As shown in Fig. 7, these two proxies are connected to each other based on the physical layer of the PCIe. The accelerator functional unit (AFU) is the core design to accelerate specific functionality, which, in this paper, is the LRCN accelerator handling the assigned layers after task partitioning. The AFU reads and writes data across the PCIe physical connection and communicates with the CPU side application behind the PSL, with the assistance of the Storage, Network, and Analytics Programming (SNAP) framework. SNAP contains a dedicated hardware IP core that communicates between the PSL and AFU in order to manage the requiring tasks initiated from the host side. The SNAP core provides a four-state finite-state machine (FSM), called CAPI FSM, to handle the job management based on the memory-mapped register I/O inputs from the host CPU. The CAPI FSM is responsible for activating the AFU execution by the communication between its control interface and the AFU's FSM (which is called accelerator FSM). The CAPI FSM contains four states including, *wait*, *assign*, *return*, and *complete*, among which the *assign* state will signify the AFU to start execution and the *return* state will initiate PSL to transmit the status information of the AFU back to the host CPU. On the host CPU side, CAPI provides CAPP as part of the PCIe Host Bridge (PHB) to handle the communications.

VII. Design Space Exploration

A. Design space definition

The proposed heterogeneous system provides us with an enormous design space to explore, covering the accelerator, host CPU, and interconnection schemes. With such a sufficiently high degree of design space, we have more opportunities to deliver optimized system-level designs for accelerating the targeted video analysis applications. We summarize the design space in Table II.

The design space outlines our proposed design by providing configurations regarding the accelerator and the CPU. It describes an adaptive space as its size is directly related to the workload partitioning scheme. Assuming the first i layers are distributed to the accelerator, the possible accelerator design combinations for AUs can reach 4^i when considering the parallel factors (*cpf*, *kpf*) and on-chip buffer setups (*InBuf*, *WBuf*). With more DNN layers allocated to the accelerator, the design space grows to contain more design dimensions. Regarding the accelerator design, we adopt three major resources as DSPs (the computation resources), BRAMs (the on-chip memory resources), and external memory access bandwidth to specify the resource constraints during design space exploration. For the other part of workloads running on the host CPU, we explore the different configurations of multithreading schemes, where the constraint is the available thread number. All the configurable parameters listed in Table II contribute to high-dimension design space, providing us a great degree of design freedom and increasing the design difficulties for choosing the most suitable design combination by considering given resource constraints and performance targets.

B. Overall DSE flow

We propose a DSE engine (*Step 3* of the EcoSys flow shown in Fig. 3) to identify suitable design configurations and deliver the customized design to meet hardware specifications and performance targets. The whole DSE flow is described by Algorithm 1. It takes the targeted DNN layer information and hardware resource constraints as inputs and generates hardware configuration guidelines for implementing the whole system and the estimated achievable performance.

TABLE II
AN ADAPTIVE DESIGN SPACE OF OUR PROPOSED SYSTEM

	FPGA-based accelerator	**Host CPU**
Parameters	*Batch size*	
	TP: task partitioning scheme	
	$cpf_1, cpf_2, \cdots cpf_i$ $kpf_1, kpf_2, \cdots kpf_i$ $InBuf_1, InBuf_2, \cdots InBuf_i$ $WBuf_1, WBuf_2, \cdots WBuf_i$	*thread*: multithreading scheme
Constraints	*FPGA available resources*	*available threads*

Algorithm 1: The proposed DSE flow

```
1  Input layer info $L = \{l_1, l_2 \cdots l_n\}$
2  Input resource constraints:
     $R_{Acc} = \{DSP, BRAM, BW\}$, $R_{CPU} = THREAD$
3  Initialize best configuration: $Config_{best} = \{\}$
4  Initialize best performance: $Perf_{best} = 0$
5  Initialize task partitioning: $TP = len(L)$
6  while $TP > 1$ do
7      $Task_{Acc}, Task_{CPU} \leftarrow$ TaskPartition($TP$)
8      Initialize accelerator configuration: $config_{Acc} = \{\}$
9      $PF \leftarrow$ CompAlloc($Task_{Acc}, R_{Acc}$)
10     $\{CPF, KPF, InB, WB\} \leftarrow$ MemAlloc($PF, R_{Acc}$)
11     $config_{Acc} \leftarrow \{CPF, KPF, InB, WB\}$
12     $Perf_{Acc} \leftarrow$ AccEval($config_{Acc}$)
13     Initialize CPU configuration: $config_{CPU} = \{\}$
14     Initialize CPU performance: $Perf_{CPU} = 0$
15     for $thread$ in range($R_{CPU}$) do
16         $p \leftarrow$ CPUEval($Task_{CPU}, thread$)
17         if $p > Perf_{CPU}$ then
18             $config_{CPU} \leftarrow thread$
19             $Perf_{CPU} \leftarrow p$
20     end
21     $Perf \leftarrow min\{Perf_{Acc}, Perf_{CPU}\}$
22     if $Perf > Perf_{best}$ then
23         $Config_{best} \leftarrow \{config_{Acc}, config_{CPU}\}$
24         $Perf_{best} \leftarrow Perf$
25     $TP = TP - 1$
26 end
27 return $Config_{best}, Perf_{best}$
```

Task partitioning is first performed in Algorithm 1 to divide the targeted workload into two halves. For a given DNN model with n major layers, there are $n-1$ partition schemes to be explored. We use TP to denote the "split point", meaning that assign layer 1 $\sim TP$ are assigned to the customized accelerator while layer $TP \sim n$ are assigned to the host CPU. From Line 6 $\sim$ 26, we start traversing all partition schemes. In the proposed DSE design, we retain an assumption that layers in the front part of DNN are more likely to have higher CTC ratios (e.g., the examples shown in Fig. 2), which are more compute-intensive and suitable to be handled by the FPGA-based accelerator compared to layers in the rear part.

For each partition scheme, individual optimizations are launched to deliver configurations for the accelerator (Line 8 $\sim$ 12) and the CPU (Line 13 $\sim$ 20) given available hardware resources. Regarding the accelerator optimization, we first adopt **CompAlloc** function to allocate the available compute resources to maximize the performance of the hardware accelerator following a pipeline architecture. These stage-wise parallel factors are then packed as PF and passed to **MemAlloc** for memory resource allocation. The utilization of on-chip memory and external memory access bandwidth is closely related to the parallel factor and DNN layer structure. In our design, the weight buffer size is determined by the assigned parallel factor and DNN kernel size, while the input buffer size is determined by the number of cached slices of the input feature map as mentioned in Sec. VI-B2. The goal of memory resource allocation is to provide timely data delivery so that the allocated compute resources can perform smoothly. In the next step, **AccEval** is launched to evaluate the current accelerator configuration.

On the host CPU side, we evaluate its performance to handle the $Task_{CPU}$ (with DNN layers assigned to the CPU) and the goal is to discover a configuration that can match the accelerator's performance (e.g., to achieve similar throughput for a better system-level pipeline). We enumerate all possible configurations of the thread number in a binary search manner when using OpenMP and collect the best one.

C. Compute resource allocation

One of the most critical problems in FPGA-based DNN implementation is unclear resource allocation. Therefore, we propose a function called **CompAlloc** to provide resource allocation guidelines by considering targeted workloads and pre-defined hardware resource constraints. As shown in Algorithm 2, we allocate the computation resource to balance the latency of each pipeline stage. Assuming the computation demand of pipeline stage i is C_i (corresponding to major layer i), the increase of allocated compute resource R_i for this stage results in a proportional increase in parallelism (meaning larger cpf_i and kpf_i) and eventually lowers the execution latency. By configuring the allocated compute resources, we can balance the DNN workloads by coordinating every pipeline stage and achieve the maximum throughput once the workload for each stage is well-balanced. Since the parallelism factors must be the power of 2, we further fine-tune the allocation scheme and fills the gap between the actual and the theoretical values.

Algorithm 2: Compute resource allocation

```
Input available compute resource: $R_{total}$
Input stage-wise compute demands: $C_i$, and total demands: $C_{total}$
Resource distributed to pipeline stage $i$: $R_i = \frac{C_i}{C_{total}} \times R_{total}$,
  $R_i = 2^{\lfloor \log_2 R_i \rfloor}$
while $\sum_{i=1}^{n} R_i \leq R_{total}$ do
    Select stage $j$ with maximum $\frac{C_j}{R_j}$: $R_j = 2 \times R_j$
    if $\sum_{i=1}^{n} R_i \leq R_{total}$ then
        Continue
    else
        $R_j = \frac{R_j}{2}$
        Break
end
$R_i = cpf_i \times kpf_i$
```

D. Memory resource allocation

According to the parallel factors, the proposed DSE then starts allocating the memory resources for each pipeline stage to guarantee sufficient data supply. In our design, the output bitwidth of the input buffer is assigned to $DW_i \times cpf_i$, while the output bitwidth of the weight buffer is $WW_i \times kpf_i \times cpf_i$, where DW_i and WW_i represent the data quantization scheme of feature maps and DNN parameters in stage i, respectively. Such a design ensures that sufficient data can be obtained in every fetch performed by the proposed CE with the desired parallelism. As mentioned in Sec. VI-B2, the input buffer of stage i only needs to keep a small subset of the input feature map ($K_i + S_i$ slices). When holding S_i more slices of input feature map (e.g., $K_i + 2S_i$ slices), the same DNN parameters can be used for one more sliding window operation with a longer life after fetching from the external memory. It means buffering a slightly larger portion of the feature map helps lower the bandwidth demands as parameters are fetched less frequently. Therefore, we can also adjust the bandwidth utilization in case the available bandwidth is not enough.

E. Accelerator performance estimation

In general, a complete FPGA design flow includes several time-consuming steps, such as RTL design, logic synthesis, place-and-route. Even with a fully optimized RTL design, the rest of the design flow can take hours to generate an implementation for performance evaluation. This extremely long development process makes the proposed DSE infeasible as every task partition scheme means hours of waiting. To provide timely performance feedback, we adopt analytical models to estimate achievable performance based on the hardware configurations so that the DSE engine can make the most suitable decision in a minute.

We take a DNN with n CONV-like layers as an example, and, regarding the i-th layer, we assume the input feature map size to be $InCh_i \times H_i \times W_i$ and the kernel size to be $OutCh_i \times InCh_i \times K_i \times K_i$. With hardware configuration $Config$, the latency Lat_i during the execution of layer i at working frequency $freq$ can be determined as:

$$Lat_i = \frac{OutCh_i \times InCh_i \times H_i \times W_i \times K_i \times K_i}{cpf_i \times kpf_i \times freq} \tag{1}$$

The throughput performance of the accelerator can be described using the following equation.

$$Throughput = \frac{Batch\ size}{max(Lat_1, Lat_2, ..., Lat_n)} \tag{2}$$

The estimated throughput performance achieved by the accelerator will be the performance target for the host CPU when handling the rest of workloads ($Task_{CPU}$). To maintain the optimized system-level performance, the CPU adopts a multi-threaded optimization on the software task to achieve comparable performance. Only when the process latency of each pipeline stage from the accelerator and the latency of the CPU process are similar can we achieve a fully pipelined flow.

To demonstrate the accuracy of our proposed analytical models, we compare the estimated performance of three DNN accelerators to their actual performance after board-level implementation and present the results in Table III. In this experiment, we prepare three DNN accelerator IPs targeting AlexNet and VGG-16 with various architecture parameter setups (such as *Batch size*, cpf, kpf, etc.). As the proposed analytical models only work for the accelerator side, we do not consider the task partition feature. For case 1, we prepare an accelerator IP for running AlexNet inference with batch size = 1 at 200MHz; for case 2, we have a larger IP instance for AlexNet with batch size = 6 at 220MHz; for the last case, we construct an accelerator for VGG-16 with batch size = 2 at 235MHz. By comparing the estimated and actual performance after implementation in Table III, we capture the average performance estimation error as 0.22%, which verifies the accuracy of our proposed analytical models.

TABLE III
PERFORMANCE ESTIMATION ERRORS

	Esti. FPS	Impl. FPS	Error (%)
Case1: AlexNet IP(S)	169.5	170	0.29
Case2: AlexNet IP(L)	1130	1126	0.36
Case3: VGG-16 IP	65	65	0.00

F. Multi-threaded optimization

After the DSE decides the software partitioned task for the CPU, we then feed the original network architectural parameters into a python-based OpenMP-enabled code generation engine with the main algorithm shown in the Algorithm 3. Within the generation process, the tool utilizes a three-layer (CONV, FC, and LSTM layers) template to iterate the partitioned network with a given threading factor for each targeted layer. We leverage the OpenMP *parallel for* optimizer with *static* scheduling to increase code parallelism across the output channel dimension. Also, we utilize the OpenMP optimizer to assign *share* memory for the input and weight buffers of the targeted layer and *private* memory for the loop indexing variables. After the OpenMP-enabled C-based source code for the software task is complete, we then calibrate the threading factor in a binary search pattern that finds the design code to reach the target latency performance asked from DSE. If such a design is satisfactory, we then integrate this callable function into the main function that also calls the FPGA accelerator with the system-level pipeline.

VIII. Experimental Results

A. Preparation work

To carry out the EcoSys that generates the optimized CPU + FPGA designs, we follow the proposed four-step design flow (Fig. 3). We first feed our proposed framework with the targeted LRCN network information, which includes the Python-based DNN definition file and corresponding DNN parameters. We then enter the targeted performance and the hardware resource constraints, which contain compute and memory resources provided by the targeted FPGA and the number of available threads in the host CPU. After performing the *Analysis* and *DSE* steps, the pre-defined RTL IPs and the OpenMP templates are configured with optimized hardware parameters listed in Table II. The RTL IPs are then passed through the FPGA design and implementation flow to construct a customized accelerator running on FPGA. At the same time, the OpenMP templates are executed on the host CPU with multithread technology. Knowing the DNN compute behaviors from the model definition file, we partition

Algorithm 3: OpenMP-enabled code generation

```
SW_FUNC ← source_code_prolog
for $L_i \rightarrow SW\_layers$ do
    layer_src ← function_prolog based on $\boldsymbol{L_i.param}$
    line_src ← " #pragma omp parallel for
        shared($\boldsymbol{L_i.param}.shared\_buf$)
        privarte($\boldsymbol{L_i.param}.private\_var$) shcedule(static)
        num_threads($\boldsymbol{L_i.param}.thread\_num$)"
    layer_src.append(line_src)
    if $L_i.type$ is fully-connected_layer then
                                              ▷ FC layer optimization
        line_src ← "for c in output_channel"
        layer_src.append(line_src)
        line_src ← "for b in batch_size"
        layer_src.append(line_src)
        line_src ← "$\boldsymbol{L_i.param}.out[b][c] \leftarrow \boldsymbol{L_i.param}.bias[c]$"
        layer_src.append(line_src)
        line_src ← "for n in input_channel"
        layer_src.append(line_src)
        line_src ← "$\boldsymbol{L_i.param}.out[b][c]+=$
            $\boldsymbol{L_i.param}.in[b][n] \times \boldsymbol{L_i.param}.w[c][n]$"
        layer_src.append(line_src)
        layer_src.append(function_epilog)
    end
    else if $L_i.type$ is convolutional_layer then
                                              ▷ CONV layer optimization
        line_src ← "for c in output_channel"
        layer_src.append(line_src)
        line_src ← "for b in batch_size"
        layer_src.append(line_src)
        line_src ← "for n in input_channel"
        layer_src.append(line_src)
        line_src ← "for h in image_height"
        layer_src.append(line_src)
        line_src ← "for w in image_width"
        layer_src.append(line_src)
        line_src ← "for i in kernel_height"
        layer_src.append(line_src)
        line_src ← "for j in kernel_width"
        layer_src.append(line_src)
        line_src ← "update out[b][c][h][w] with input[b][c][h+i][w+j]
         and weight[c][n][i][j]"
        layer_src.append(line_src)
    end
    else if $L_i.type$ is LSTM_layer then
                                              ▷ LSTM layer optimization
        line_src ← "for c in output_channel"
        layer_src.append(line_src)
        line_src ← "for b in batch_size"
        layer_src.append(line_src)
        line_src ← "update out[b][c] with input[b][c] and c_tml[b][c]"
        layer_src.append(line_src)
    end
    SW_FUNC.append(layer_src)
end
SW_FUNC.append(source_code_epilog)
return SW_FUNC
```

and reorder the DNN parameters to be compatible with the hardware design. We then apply the CAPI integration and combine the FPGA accelerator and the host CPU to perform a system-level pipeline shown in Fig. 4 (b).

During the experiments, we select the Alpha Data ADM-PCIE-9H3 acceleration board with a Xilinx VU35P FPGA as our target FPGA platform for board-level implementations of the customized accelerator design. To complete the FPGA design flow, we adopt Vivado 2020.1 to synthesize and implement the proposed design at 250MHz working frequency. We adopt an IBM Power9 CPU (with 2 GHz working frequency) as the host CPU.

TABLE IV
DATA MOVEMENT OVERHEAD FOR PROCESSING EVERY VIDEO FRAME

Test case		non-CAPI scheme (ms)	CAPI scheme (our design) (ms)	Avg. latency reduction
1	HW	53.34	53.33	19.96%
	Total	163.51	130.88	
2	HW	53.39	53.41	11.72%
	Total	148.18	130.82	
3	HW	53.41	53.44	10.35%
	Total	146.23	131.09	

TABLE V
RESOURCE UTILIZATION OF THE PROPOSED LRCN ACCELERATORS

	Backbone	DSP	BRAM	LUT	FF
1	AlexNet	1882	630	218095	124286
2	VGG-16	4348	1142	397314	157788

TABLE VI
PERFORMANCE OF THE PROPOSED LRCN ACCELERATORS

	Backbone	Max. FPGA stage	CPU lat.	Batch	FPS
1	AlexNet	9.4 ms	12.7 ms	4	314.7
2	VGG-16	14.5 ms	17.2 ms	1	58.1

B. CAPI integration benefits

In the traditional CPU + FPGA system, the FPGA can only access its own memory, which locates on the FPGA board. The data of interests, such as the input video frames, and updated DNN parameters, is commonly stored inside the main memory accessible from the CPU. It means an extra data movement is inevitable from the main memory to the FPGA's own memory before activating the FPGA accelerator. The latency caused by such an extra data movement will significantly affect the overall performance when accelerating the video analysis applications. To eliminate the data movement latency between devices, we adopt a CAPI-based solution by providing a coherent memory space for both devices with equal accessibility. To demonstrate the benefit of using CAPI, we perform three experiments corresponding to test case 1 $\sim$ 3 in Table IV and provide quantitative analysis regarding the differences between CAPI and non-CAPI (traditional) solution.

We first prepare two systems for comparison as one uses traditional CPU and FPGA connection (non-CAPI), and the other uses a CAPI-connected scheme with coherent memory space. By targeting the same LRCN model with an identical task partitioning scheme, we deploy the same accelerator IP to these two systems. For test case 1, we perform continuous single frame execution to simulate most real-life cases, where video frames are transferred into the system one after another and processed by FPGA and CPU following the task partitioning scheme. For cases 2 and 3, we increase the granularity of execution and allow 10 and 100 frames to be transferred each time. Latency results of all three cases are shown in Table IV, where "HW" indicates the average process latency spending on FPGA, while "Total" means the total average execution time for the whole system for each input frame. Note that the frame collecting time is not included in these experiments, which means we assume all frames (e.g., all 10 frames in case 2, and all 100 frames in case 3) are initially available in the main memory.

In this experiment, we try to maintain the same performance from the hardware accelerator side in both CAPI and non-CAPI systems and mainly focus on the differences caused by data movements. Results in Table IV demonstrate that our proposed design (with CAPI) can reduce up to 19.96% of latency in test case 1 compared to the traditional scheme. The latency advantage of our design sightly decreases to 11.72% and 10.35% of latency reduction for cases 2 and 3, respectively.

C. The baseline designs

We select our previous LRCN accelerator design in [25] as one of our baseline designs (named baseline 1). It is implemented on a VC709 FPGA with an older technology node (28 nm) compared to our targeted FPGA (16 nm) in this work. To have a fair comparison, we create the second baseline design (baseline 2) by implementing the previous design on the same VU35P FPGA. We still maintain the same features adopted by the previous design, such as using the same configurable HLS IPs and the same memory hierarchical design. The performance of these two baseline designs is shown in Table VII.

D. The EcoSys proposed designs

By following the EcoSys design flow, we generate two customized heterogeneous system designs for accelerating LRCN with AlexNet and VGG-16 backbones. Although these two LRCN models have different front-end CNNs, the rest of these models are identical, which adopt the same LSTM layers with 15 iterations. For the LRCN with AlexNet backbone, we adopt the unpruned version with 5 CONV and 3 FC layers with 1.45 GOP compute complexity. According to the task partition scheme generated by EcoSys, the whole AlexNet backbone is mapped to the targeted FPGA while the LSTM layers are handled by CPU. We use the original VGG-16 as the backbone for the second design, which contains 13 CONV and 3 FC layers with 30.9 GOP. In this case, CONV layers are distributed to the FPGA while the FC and LSTM layers are assigned to the CPU.

To improve hardware efficiency, we use fixed16 precision to represent AlexNet's activations and parameters and VGG-16's activations while use fixed8 precision to represent parameters in VGG-16. Regarding the layers mapped to the CPU, we use float32 precision. The FPGA-based accelerators in both designs are implemented with 250MHz working frequency, and resource utilization is presented in Table V. We also present the parallel factor configurations of the proposed accelerators in Fig. 8. These configurations are generated by the proposed DSE, which helps create balanced pipeline stages for optimized overall performance.

Performance results are presented in Table VI. As we adopt a layer-based pipeline architecture, the performance of the hardware accelerator is limited by the slowest pipeline stage. Latency results of these critical stages are presented in the "Max. FPGA stage" column. On the other side, the host CPU contributes to the last pipeline stage by handling the rest of the workloads. So, the overall throughput of the proposed LRCN accelerator is 314.7 FPS (with AlexNet) and 58.1 FPS (with VGG-16).

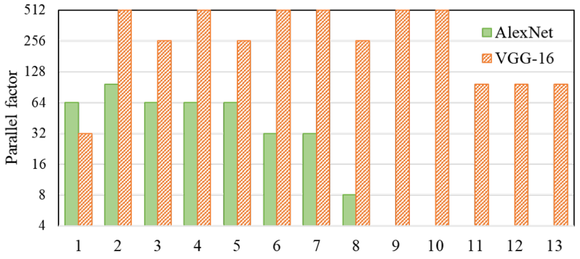


Fig. 8. Parallel factor configuration of the proposed designs. We label the layers along with the x-axis. For the design using AlexNet backbone, layer 1 ∼ 8 indicate 5 CONV and 3 FC layers; while for the VGG-backbone, layer 1 ∼ 13 indicate all CONV layers in VGG-16. (Pooling layers are omitted)

E. Comparison results

We compare the proposed design to other implementations using pure CPU and FPGA by targeting the same LRCN model with AlexNet backbone. For the CPU design, we use the same Power9 CPU to execute the whole LRCN and adopt the proposed OpenMP templates for implementing all DNN layers. We list the performance of two CPU designs in Table VII. To search for the best thread number configuration, we adopt a system-level profiling tool called OProfile and eventually conclude that using 32 threads can reach the optimal performance for the targeted workloads, peaking at 15.1 FPS. We also include two FPGA designs as baselines. Compared to the multithreaded CPU and FPGA baseline 2, our proposed design achieves 20.8× and 5.3× higher throughput, respectively. Such a significant performance improvement demonstrates the effectiveness of using proposed optimization methods.

We also extend the comparison to other customized accelerator design frameworks for accelerating DNN inference. Since these frameworks are not originally built for video analysis, they solely focus on CNNs but not CNN + RNN structures. To enable a suitable comparison, we disable EcoSys's task partition feature and map the targeted CNN only into FPGA for performance evaluation. Results are shown in Table VIII. The target DNN is VGG-16 [4], which costs 30.9 Giga operations for processing one input image. Our proposed VGG-16 accelerator adopts a fixed16 precision for activation and a fixed8 precision for DNN parameters. After implementing on the targeted FPGA, the proposed VGG-16 accelerator consumes 4410 DSPs, 1293 Block RAMs, 415252 LUTs, and 170033 Flip Flops. In this table, we report the actual DSP utilization for 16-bit data multipliers. For the design using Intel FPGAs published (e.g., [39]), the DSP utilization should be twice as shown in [39] because one variable-precision DSP block can support two 16-bit multipliers at the same time. By comparing the throughput performance, our proposed design can reach 2141.0 GOP/s with batch size 2, which is 1.17× and 1.42× higher than the second and third best designs.

TABLE VII
RESULT COMPARISON FOR RUNNING LRCN WITH ALEXNET BACKBONE

Platform	Freq. (MHz)	FPS
Power9 CPU (1 thread)	2000	0.8
Power9 CPU (32 threads)	2000	15.1
VC709 FPGA Baseline1 [25]	100	25.0
VU35P FPGA Baseline2	250	59.1
The proposed design	CPU: 2000 FPGA: 250	314.7

TABLE VIII
PERFORMANCE COMPARISON OF ACCELERATOR DESIGN FRAMEWORKS

	[39]	[23]	[12]	Ours
Model	VGG-16			
Device	GX1150	VU9P	VU9P	VU35P
Tech. node	20 nm	16 nm	16 nm	16 nm
Freq.	200MHz	210MHz	214MHz	250MHz
Used DSP	3036	4096	5349	4410
GOP/s	720.2	1510.0	1828.6	**2141.0**
GOP/s/DSP	0.24	0.37	0.34	**0.49**

IX. CONCLUSIONS

This paper presented EcoSys, a novel framework to explore co-optimization opportunities for building customized heterogeneous systems with both CPU and FPGA-based DNN accelerator for video analysis workloads. To the best of our knowledge, it is the first hardware-software co-optimization framework that supports a CAPI-connected CPU + FPGA system and provides an end-to-end design and optimization flow from DNN design to its hardware implementation on the heterogeneous system. To address the design challenges, we adopted a CAPI-enable coherent memory space to connect between the accelerator and the host CPU, which saves up to 19.96% of data transfer latency compared to the traditional CPU + FPGA system. We proposed a highly configurable accelerator architecture and diverse optimization strategies, including layer-based pipeline, efficient memory hierarchical design, high-performance RTL layer IPs for the accelerator, and multithreaded layer templates for the CPU to satisfy demanding workloads. We then introduced an adaptive design space to describe the possible hardware configurations and a DSE engine to explore optimized solutions by considering various hardware constraints and performance targets. With the above novel designs, EcoSys achieved 20.8× and 5.3× higher throughput compared to the multithreaded CPU and pure FPGA designs when targeting the same LRCN model.

ACKNOWLEDGEMENTS

This work is supported in part by the IBM-Illinois Center for Cognitive Computing System Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. This work utilizes resources supported by the National Science Foundation's Major Research Instrumentation program, grant #1725729, as well as the University of Illinois at Urbana-Champaign. Xiaofan Zhang is supported by a Google PhD Fellowship.

REFERENCES

[1] J. o. Donahue, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the conference on computer vision and pattern recognition (CVPR)*, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the Advances in neural information processing systems*, 2012.

[3] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2014.

[4] K. Simonyan *et al.*, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[5] K. He *et al.*, "Deep residual learning for image recognition," in *Proc. of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

[6] J. Qiu *et al.*, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016.

[7] X. Zhang *et al.*, "DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2018.

[8] N. Suda *et al.*, "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016.

[9] S. Han *et al.*, "ESE: Efficient speech recognition engine with sparse LSTM on FPGA," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[10] K. Wakabayashi, "System LSI design with c-based behavioral synthesis and verification," in *Proceedings of International Symposium on Circuits and Systems (ISCAS)*, 2005.

[11] K. Rupnow *et al.*, "High level synthesis of stereo matching: Productivity, performance, and software constraints," in *Proceedings of the International Conference on Field-Programmable Technology (FPT)*, 2011.

[12] Y. Chen *et al.*, "Cloud-DNN: An open framework for mapping DNN models to cloud FPGAs," in *Proc. of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2019.

[13] H. Ye *et al.*, "HybridDNN: A framework for high-performance hybrid DNN accelerator design and implementation," in *Proceedings of the Design Automation Conference (DAC)*, 2020.

[14] Y. Zhang *et al.*, "FracBNN: accurate and FPGA-efficient binary neural networks with fractional activations," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[15] C. Hao *et al.*, "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proceedings of the Design Automation Conference (DAC)*, 2019.

[16] X. Zhang *et al.*, "SkyNet: a hardware-efficient method for object detection and tracking on embedded systems," in *Proceedings of the Conference on Machine Learning and Systems (MLSys)*, 2020.

[17] Q. Huang *et al.*, "CoDeNet: Efficient deployment of input-adaptive object detection on embedded FPGAs," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2021.

[18] Q. Li *et al.*, "Implementing neural machine translation with bi-directional GRU and attention mechanism on FPGAs using HLS," in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2019.

[19] B. Li *et al.*, "FTRANS: energy-efficient acceleration of transformers using FPGA," in *Proc. of International Symposium on Low Power Electronics and Design (ISLPED)*, 2020.

[20] Q. Li *et al.*, "Efficient methods for mapping neural machine translator on FPGAs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1866–1877, 2021.

[21] C. Zhang *et al.*, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.

[22] 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.

[23] X. Wei *et al.*, "TGPA: tile-grained pipeline architecture for low latency cnn inference," in *Proceedings of the International Conference on Computer-Aided Design (ICCAD)*, 2018.

[24] C. Zhang *et al.*, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2015.

[25] X. Zhang *et al.*, "High-performance video content recognition with long-term recurrent convolutional network for FPGA," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[26] R. Zhao *et al.*, "Accelerating binarized convolutional neural networks with software-programmable fpgas," in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2017.

[27] J. Wang *et al.*, "Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA," in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, 2018.

[28] Q. Xiao *et al.*, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in *Proceedings of the Design Automation Conference (DAC)*, 2017.

[29] C. Zhuge *et al.*, "Face recognition with hybrid efficient convolution algorithms on FPGAs," in *Proceedings of the Great Lakes Symposium on VLSI (GLSVLSI)*, 2018.

[30] X. Wei *et al.*, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the Design Automation Conference (DAC)*, 2017.

[31] Y. Guan *et al.*, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017.

[32] X. Zhang *et al.*, "DNNExplorer: A framework for modeling and exploring a novel paradigm of FPGA-based DNN accelerator," in *Proceedings of the International Conference On Computer Aided Design (ICCAD)*, 2020.

[33] L. Liao *et al.*, "UHCL-darknet: An OpenCL-based deep neural network framework for heterogeneous multi-/many-core clusters," in *Proceedings of International Conference on Parallel Processing (ICPP)*, 2018.

[34] S. Zheng *et al.*, "Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

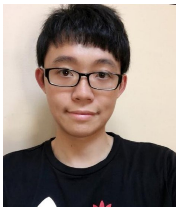
[35] W. Jiang *et al.*, "Accuracy vs. efficiency: Achieving both through FPGA-implementation aware neural architecture search," in *Proc. of Design Automation Conference (DAC)*, 2019.

[36] ——, "Hardware/software co-exploration of neural architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4805–4815, 2020.

[37] Y. Li *et al.*, "EDD: Efficient differentiable DNN architecture and implementation co-search for embedded AI solutions," in *Proc. of Design Automation Conference (DAC)*, 2020.

[38] J. Stuecheli, B. Blaner, C. R. Johns, and M. S. Siegel, "CAPI: A Coherent Accelerator Processor Interface," *IBM J. Research and Develop.*, vol. 59, no. 1, pp. 7:1–7:7, Jan 2015.

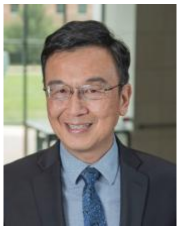
[39] Y. Ma *et al.*, "An automatic RTL compiler for high-throughput FPGA implementation of diverse deep convolutional neural networks," in *International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–8.



Xiaofan Zhang received the B.S. and M.S. degrees from the University of Electronic Science and Technology of China in 2013 and 2016, respectively. He is a Ph.D. Candidate with the ECE Department, UIUC. His research interests include deep learning accelerator design, hardware-software co-design, and energy-efficient computing. He has received the IEEE/ACM William J. Mccalla ICCAD Best Paper Award in 2018, the IEEE/ACM Design Automation Conference System Design Contest double championships in 2019, and the Google Ph.D. Fellowship in 2020.

Yuan Ma received the B.S. degree in Electrical and Computer Engineering from the University of Illinois at Urbana–Champaign (UIUC) in 2020. He is a Ph.D. student with the ECE department, UIUC. He is also a research assistance in the Center for Artificial Intelligence Innovation at National Center for Supercomputing Applications (NCSA).

Jinjun Xiong received the Ph.D. degree from the University of California, Los Angeles, CA, USA, in 2006. He is a Research Staff Member and Program Director for cognitive computing systems research with the IBM T.J. Watson Research Center. He co-founded and co-directs the IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR). He is also an adjunct Research Professor with the ECE Department, UIUC. His research interests include AI, Machine Learning and Systems. His research was recognized with seven best paper awards, three Best Poster Awards, and eight nominations for best paper awards at various international conferences. He also won numerous top awards at various international competitions in AI and systems optimization.



Wen-Mei W. Hwu joined NVIDIA in February 2020 as Senior Distinguished Research Scientist, after spending 32 years at the UIUC, where he was a Professor, Sanders-AMD Endowed Chair, Acting Department Head and Chief Scientist of the Parallel Computing Institute. For his research contributions, Hwu received the ACM SigArch Maurice Wilkes Award, the ACM Grace Murray Hopper Award, the IEEE Computer Society Charles Babbage Award, the ISCA Influential Paper Award, the MICRO Test-of-Time Award, the IEEE Computer Society B. R. Rau Award, the CGO Test-of-Time Award and the Distinguished Alumni Award in CS of the University of California, Berkeley. He has also won numerous best paper awards for major conferences. He is a Fellow of IEEE and ACM.

Volodymyr Kindratenko received the Specialist degree from the Vynnychenko State Pedagogical University, Kirovograd, Ukraine, in 1993, and the D.Sc. degree from the University of Antwerp, Belgium, in 1997. He is a Senior Research Scientist at the National Center for Supercomputing Applications (NCSA), and Adjunct Associate Professor at the ECE Department and a Research Associate Professor at the CS Department, UIUC. At NCSA, he is the Director of Innovative Systems Lab an a co-director of the Center for AI Innovation. His research interests include high-performance computing and machine learning with special-purpose architectures. He is a Senior Member of IEEE and ACM, and a Novel Architectures department editor for Computing in Science and Engineering magazine.

Deming Chen received the B.S. degree in Computer Science from the University of Pittsburgh in 1995, and the M.S. and Ph.D. degrees in computer science from the University of California at Los Angeles in 2001 and 2005, respectively. He is the Abel Bliss Endowed Professor with the ECE Department, UIUC. His research interests include system-level and high-level synthesis, machine learning, computational genomics, GPU and reconfigurable computing, and hardware security. He has received the UIUC's Arnold O. Beckman Research Award, the NSF CAREER Award, nine best paper awards, the ACM SIGDA Outstanding New Faculty Award, and the IBM Faculty Award. He is an IEEE Fellow, an ACM Distinguished Speaker, and the Editor-in-Chief of ACM Transactions on Reconfigurable Technology and Systems (TRETS).