Remarn: A Reconfigurable Multi-threaded Multi-core Accelerator for Recurrent Neural Networks

ZHIQIANG QUE, Imperial College London, UK
HIROKI NAKAHARA, Tokyo Institute of Technology, Japan
HONGXIANG FAN, Imperial College London, UK
HE LI, University of Cambridge, UK
JIUXI MENG, Imperial College London, UK
KUEN HUNG TSOI and XINYU NIU, Corerain Technologies Ltd., China
ERIKO NURVITADHI, Intel Corporation, USA
WAYNE LUK, Imperial College London, UK

This work introduces Remarn, a reconfigurable multi-threaded multi-core accelerator supporting both spatial and temporal co-execution of Recurrent Neural Network (RNN) inferences. It increases processing capabilities and quality of service of cloud-based neural processing units (NPUs) by improving their hardware utilization and by reducing design latency, with two innovations. First, a custom coarse-grained multi-threaded RNN/Long Short-Term Memory (LSTM) hardware architecture, switching tasks among threads when RNN computational engines meet data hazards. Second, the partitioning of this hardware architecture into multiple full-fledged sub-accelerator cores, enabling spatially co-execution of multiple RNN/LSTM inferences. These innovations improve the exploitation of the available parallelism to increase runtime hardware utilization and boost design throughput. Evaluation results show that a dual-threaded quad-core Remarn NPU achieves 2.91 times higher performance while only occupying 5.0% more area than a single-threaded one on a Stratix 10 FPGA. When compared with a Tesla V100 GPU implementation, our design achieves 6.5 times better performance and 15.6 times higher power efficiency, showing that our approach contributes to high performance and energy-efficient FPGA-based multi-RNN inference designs for datacenters.

CCS Concepts: • **Hardware → Hardware accelerators**; **Application specific processors**; • **Computer systems organization → Multicore architectures**; **Neural networks**; **Special purpose systems;**

Additional Key Words and Phrases: Accelerator architecture, recurrent neural networks, multi-tenant execution

The support of the United Kingdom EPSRC (grant numbers EP/V028251/1, EP/L016796/1, EP/N031768/1, EP/P010040/1, and EP/S030069/1), Corerain and Intel is gratefully acknowledged.
Authors' addresses: Z. Que, H. Fan, J. Meng, and W. Luk, Imperial College London, Exhibition Rd, South Kensington, London SW7 2BX, UK; emails: {z.que, h.fan17, jiuxi.meng16, w.luk}@imperial.ac.uk; H. Nakahara, Tokyo Institute of Technology, Ohokayama 1-21-2, Tokyo, 1528550, Japan; email: nakahara.h.ad@m.titech.ac.jp; H. Li, University of Cambridge, Cambridge CB2 1TN, UK; email: he.li@ieee.org; K. H. Tsoi and X. Niu, Corerain Technologies Ltd.,14F Changfu Jinmao Building (CFC), Shenzhou, China; email: {kuenhung.tsoi, xinyu.niu}@corerain.com; E. Nurvitadhi, Intel Corporation, Jones Farm Campus, Hillsboro, OR, 97124-6463, USA; email: eriko.nurvitadhi@intel.com.

1936-7406/2022/12-ART4 $15.00
https://doi.org/10.1145/3534969

ACM Reference format:
Zhiqiang Que, Hiroki Nakahara, Hongxiang Fan, He Li, Jiuxi Meng, Kuen Hung Tsoi, Xinyu Niu, Eriko Nurvitadhi, and Wayne Luk. 2022. Remarn: A Reconfigurable Multi-threaded Multi-core Accelerator for Recurrent Neural Networks. *ACM Trans. Reconfig. Technol. Syst.* 16, 1, Article 4 (December 2022), 26 pages.
https://doi.org/10.1145/3534969

1 INTRODUCTION

Recurrent Neural Network (RNN) has been the key component of **artificial intelligence (AI)** applications where the inputs are sequential, such as natural language processing [25], speech recognition [2, 28], and video analysis [17, 87]. **Long Short-Term Memory (LSTM)** is the most popular type of RNNs. Since low latency is key for a seamless user experience in such applications, efficient and real-time acceleration of RNNs/LSTMs is required. FPGAs have been used to speed up the inference of LSTMs [19, 27, 51, 52, 75], showing the benefits of low latency and low power consumption compared to CPUs or GPUs.

However, existing RNN/LSTM accelerators cannot support cost-effective multi-RNN execution. Cloud providers must minimize their huge operation costs by running as many applications on a given server as possible, while satisfying the quality of each service. In Google data centers, **Convolutional Neural Networks (CNNs)** and **Multi-Layer Perceptions (MLP)** comprise 5% and 61% of the workload, respectively, while LSTMs makes up 29% [35]. However, most of the existing LSTM accelerators are only able to perform one inference at a time [19, 27, 28, 47, 51, 57]. These accelerators can process multi-LSTM tasks by executing one by one in sequence, resulting in inefficiency when multiple requests come at the same time as shown in Figure 1. It may make later tasks wait for a long time before a hardware core is available, since earlier LSTM tasks may have a large number of timesteps involving many iterations, e.g., an LSTM layer in DeepSpeech [29] has 1,500 timesteps. Besides, some applications employ not one but multiple LSTMs to collaboratively achieve satisfactory performance. A spacecraft anomalies detection system [31] even contains over hundreds of LSTM models, each modeling a single telemetry channel and predicting values for that channel, which demonstrates the necessity of supporting multi-LSTM execution. Furthermore, conventional LSTM accelerators are often implemented by deploying all computing resources to support a single computational engine on a large scale, leveraging data-level parallelism. For instance, Brainwave [19] devised by Microsoft is a single-threaded **neural processing unit (NPU)** that involves 96,000 **processing elements (PEs)**. However, when the workload of a targeted LSTM task is small, these hardware resources will not be fully utilized, e.g., the hardware utilization is lower than 1% [19] for Brainwave and lower than 15% for the Brainwave-like NPU [51] when running an LSTM model (h_t=256). It is challenging to design an accelerator to support cost-effective multi-LSTM execution.

This article introduces a reconfigurable multi-threaded multi-core NPU for accelerating RNN/LSTM inferences by increasing the hardware utilization for better performance. It improves the processing abilities of cloud-based NPUs as well as the Quality of Service. Our primary goal is to efficiently enhance the scalability potential of NPU cores. The most area- /cost-efficient way to add logical cores is multithreading. Essentially, multithreading retrieves unused performance (where computational cores are idle because of events) by switching to another thread. Multithreading also does not affect peak performance when working in a single-threaded mode. Usually, the execution of multiple neural networks has the potential to mitigate the idle issues, because layers from different neural networks can be scheduled freely without any issue of dependencies. Running multiple tasks can also be realized by batch techniques that provide multiple requests to a neural network to produce multiple results together. However, the batch techniques can harm latency,

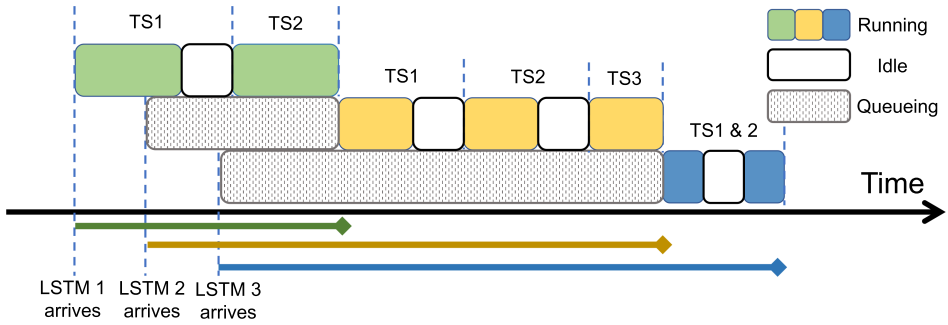

Fig. 1. Timeline of three LSTM inference tasks based on a first-come first-serve scheduling policy. The example of the LSTM-1 has two timesteps (TS), the LSTM-2 has three timesteps and the LSTM-3 has two timesteps in this figure.

because different requests may not arrive at the same time [22], which means that a newly arrived request must wait until the batch is formed, which brings a significant latency penalty as shown in Figure 2.

Remarn is inspired by coarse-grained multithreading utilized in modern CPUs such as IBM RS64-IV [6] and Intel Montecito [46], which makes a core switch to a different hardware context when a thread is stalled due to some events. Remarn consists of a custom **coarse-grained multi-threaded (CGMT)** LSTM hardware architecture that switches tasks among threads when LSTM computational engines meet data hazard. When one logical NPU core is stalled, the other can make progress. Coarse-grained multithreading is a mature technique in modern CPU designs. However, few studies concern combining the CGMT and NPUs, especially for RNNs/LSTMs. There is also fine-grained multi-threading [73, 77] that switches the context every cycle, but it brings more hardware complexity than CGMT. Unlike CNNs that do not have memory cells and can run different layers from different neural networks iteratively, RNNs/LSTMs contains many memory cells, which makes it difficult to process different timesteps from different RNN layers or models, since they have different cell memory statuses when using a single-threaded NPU. It has to finish the running of the preceding RNN inference or layer until the next inference or layer can start. The existence of inter-timestep dependencies within an RNN model prevents the following timesteps from even starting their execution until the current timestep's completion, leading to hardware underutilization and inefficiency. To address this challenge, this work proposes the CGMT-LSTM, which can intelligently switch to an alternate thread of computation when the data hazard happens, e.g., the inter-timestep dependencies of RNNs, to increase hardware utilization and boost design performance.

Besides, instead of deploying all computing resources to form a single physical core on a large scale like Brainwave [19] and our previous design [59], we design an accelerator hardware architecture that can be partitioned into multiple full-fledged sub-accelerator cores, and each core can accept new RNN requests. The multiple LSTM models used in the application of spacecraft anomaly detection system [31] have only a small hidden vector size that is less than 128, but they have 250 timesteps that are large. LSTM models that have a large number of timesteps but utilize a small size are the most tangible examples, since they require dealing with lots of dependencies, as well as the parallel task of **matrix-vector multiplications (MVMs)** [86]. The Brainwave [19] involves big MVM "tile engines" that can effectively process a 400×240 matrix in parallel. Besides, our previous NPU [59] is based on a tile size of $1,024 \times 16$, resulting in 16,384 effective PEs. Any MVM that does not map to this dimension will leave some resources idle and small MVMs even

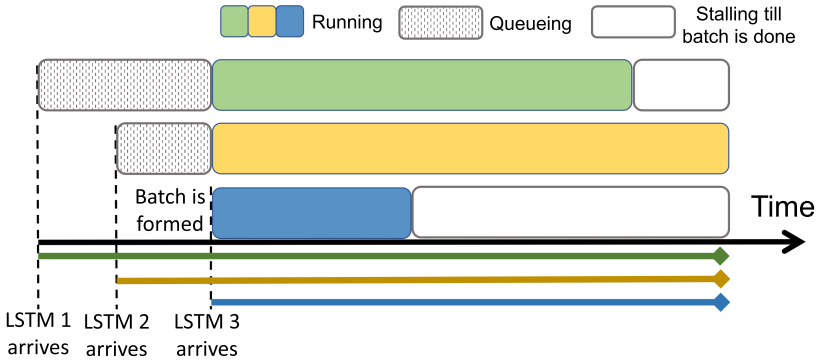

Fig. 2. Timeline of three LSTM inference tasks using a batching technique.

require zero paddings, resulting in low utilization. Splitting a large accelerator into several small sub-accelerator cores can not only help to find a better mapping to improve hardware utilization but also provide more physical cores to spatially co-locate multiple RNN inferences on the same hardware. As Reference [19] mentions, RNN programs have a critical loop-carry dependence on the h_t vector. If the full pipeline cannot return h_t to the vector register file in time for the next timestep computation, then the MVM unit will stall until h_t is available. Thus, even if a large MVM engine finishes the LSTM gates operations in a short period, e.g., one cycle, then the design still needs to wait for the h_t returned from the pipeline, which shows the limitation of the architecture using a large engine. This work splits the accelerator with a single large engine into several smaller sub-accelerator cores, each of them processing small LSTM models more efficiently by adopting the tiling-based columnwise MVM approach [57]. It addresses the challenge of accelerating a large number of small LSTM models with large timesteps. Please note that these sub-accelerator cores can work together as a single large accelerator when necessary to deal with the high priority workloads that require the lowest end to end latency.

To the best of our knowledge, Remarn is the first coarse-grained multi-threaded and multi-core LSTM accelerator architecture capable of achieving high performance and efficient multi-LSTM execution.

Our contributions are the following:

- A novel reconfigurable multi-threaded multi-core neural processing unit to enable effective and efficient multi-neural network execution for LSTM inferences.
- A custom CGMT LSTM accelerator architecture that also can be partitioned into several full-fledged sub-accelerator cores, which significantly improves hardware utilization and design performance.
- A custom tiling method for LSTMs, which minimizes the intermediate results buffers when combining CGMT and partition, thereby increasing the accelerator area efficiency.
- A comprehensive evaluation on the proposed methods and hardware architecture.

Relationship to Prior Publications: This article expands on a conference paper [59] with the baseline design proposed in Reference [57]. The baseline design [57] involves a novel latency-hiding hardware architecture based on columnwise matrix-vector multiplication. It has much higher performance and hardware utilization than other designs for RNN models with different sizes, but it still suffers from underutilization when the model size is small. Reference [59] addresses the underutilization issue by introducing CGMT that enables temporal multi-neural network execution to improve the performance when the RNN models are small while still maintaining the

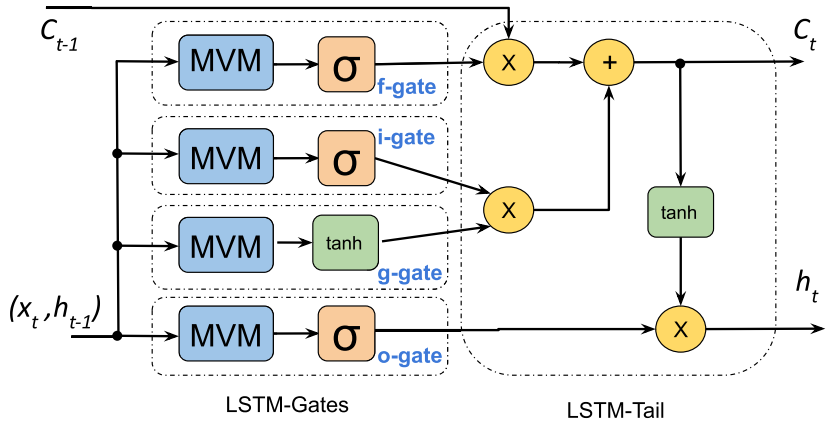

Fig. 3. A diagram of an LSTM Cell.

performance of RNN models with medium to large sizes. It mainly time-multiplexes a large RNN accelerator across various RNNs workloads. This article addresses a limitation of the work described in our previous papers, that they do not allow the large computation engine to be partitioned into several smaller ones, so they do not support spatial co-execution of multiple RNN inferences. This limitation brings a severe hardware underutilization issue when targeting acceleration for a large number of small RNN models that are commonly used in many applications [27, 31, 62, 88]. This work introduces an accelerator hardware architecture that can be partitioned into multiple full-fledged sub-accelerator cores, combining multithreading with multi-core techniques to enable temporal and spatial multi-neural network execution on cloud-based NPUs. The proposed novel dimension of optimization allows us to obtain significant improvement in throughput while reducing latency over the previous design [59]. This article adds the design of multiple sub-accelerator cores for spatial co-execution of RNNs in Section 3.3. Sections 4.1–4.3 describe a revised hardware architecture, and Sections 5.3–5.6 contain new evaluation results.

2 BACKGROUND AND PRELIMINARIES

RNNs/LSTMs have been shown to have useful properties with many significant applications. Among the many RNN variants, the most popular one is the LSTM that was initially proposed in 1997 [30]. This study follows the standard LSTM cell [17, 19, 27, 51]. Figure 3 contains a diagram of an LSTM cell. It utilizes the following equations to compute the gates and produce the results for the next time step.

$$\begin{aligned} i_t &= \sigma(W_i[x_t, h_{t-1}] + b_i), & f_t &= \sigma(W_f[x_t, h_{t-1}] + b_f) \\ g_t &= \tanh(W_g[x_t, h_{t-1}] + b_u), & o_t &= \sigma(W_o[x_t, h_{t-1}] + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t, & h_t &= o_t \odot tanh(c_t). \end{aligned} \tag{1}$$

Here, σ and $tanh$ represent the sigmoid function and hyperbolic tangent function. Both are activation functions; i_t, f_t, g_t, and o_t denote the output of the input gate (i-gate), forget gate (f-gate), input modulation gate (g-gate), and output gate (o-gate) at timestep t, respectively. The g-gate is often considered as a sub-part of the i-gate. Each LSTM gate consists of a MVM unit and a corresponding activation function unit, as shown in Figure 3. The $\odot$ operator denotes an element-wise multiplication. W is the weight matrix for both input and hidden units, since the input vector

Table 1. System Parameters

W	Weights matrix	Lh	The number of elements in a hidden vector h
Hw	The number of columns of weight matrix	TS	Timestep
Lw	The number of rows of weight matrix	NPE	The number of processing elements
x_t	The input vector x at timestep t	EP	Element-based Parallelism
h_t	The hidden vector h at timestep t	VP	Vector-based Parallelism
Lx	The number of elements in an input vector x	N	The number of sub-accelerator cores

and hidden vector are combined in the equations. The b terms denote the bias vectors. c_t is the internal memory cell status at timestep t while h_t is the hidden vector that is the output of the cell and passed to the next timestep calculation or next LSTM layer.

The LSTM information flow is controlled by these four gates with details shown in Figure 3. The i-gate decides what new information is to be written into the internal memory cell; the g-gate modulates the information processed by i-gate via adding non-linearity. Note that only g-gate utilizes hyperbolic tangent as its activation function while all the other three gates utilize sigmoid. The f-gate decides what old information is no longer needed and can be discarded so there are element-wise multiplications between the output of f-gate and memory cell status in the previous timestep c_{t-1}. Its output will be added to the products of the outputs from i-gate and g-gate to form the current status of the internal memory cell. The o-gate decides what the value of the current hidden vector (h_t) should be by multiplying the current status of the memory cell after the hyperbolic tangent function, as shown in the LSTM-Tail in Figure 3. Our work focuses on the optimization of RNN inferences involving standard LSTMs, but the proposed techniques can be applied to other **deep neural networks (DNN)** inferences.

3 DESIGN AND OPTIMIZATION METHODOLOGY

Accelerating RNN/LSTM inferences efficiently is challenging because of their recurrent nature and data dependencies. This section first presents the coarse-grained multithreading for accelerating RNN/LSTM and then introduces a partitioning strategy of LSTM weight matrix to apply sub-layer granularity scheduling with fine-grained tasks for Remarn architecture. Finally, we present the multi-core accelerator to enable spatial co-execution of RNN models or layers to improve the design performance. Some design parameters are defined in Table 1.

3.1 Multithreading for Recurrent Neural Processing Unit

There is a large demand for architectural support of multi-DNN execution to maximize the hardware utilization and reduce the costs of running a large-scale production system. However, most of the existing LSTM accelerators can only run one task at a time [19, 27, 28, 47, 51, 57]. This work proposes a CGMT LSTM NPU that switches on the event when the data hazard of a computational unit happens in the LSTM computation. The general idea is when a thread is stalled because of some events, e.g., cache misses, the multi-threaded core can switch to a different hardware context. In the proposed CGMT LSTM accelerator, the event is the hazard caused by the data dependency between the timesteps of sequential calculation in LSTMs. We propose to maintain multiple thread contexts in a recurrent NPU core so that when the first thread stalls, the second one can carry on, as shown in Figure 4. Thus, it utilizes computational resources more efficiently than a single thread core. It can increase the design performance by utilizing thread-level parallelism and enhance the NPU hardware utilization.

Besides, a preceding LSTM model or layer may have thousands of sequences (timesteps), which occupies the NPU core for a long time, leading to a long waiting time for the latter requests before they have an available core to run. However, some services have strict latency constraints, since

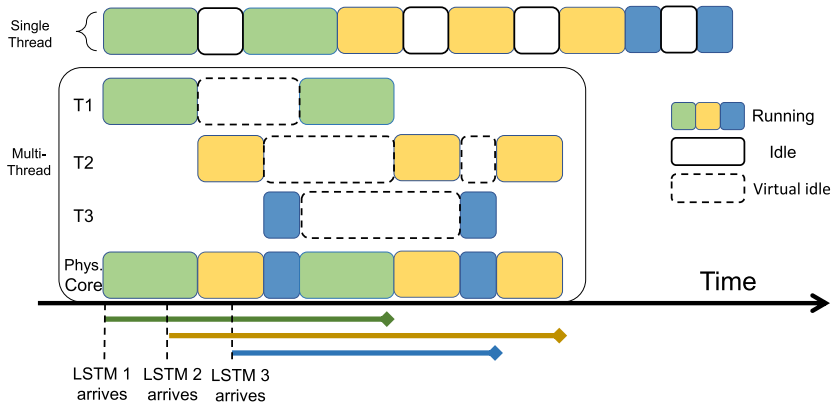

Fig. 4. Timeline of three LSTM tasks based on the proposed coarse-grained multithreading.

the response time directly relates to user experience, e.g., intelligent personal assistants are one of the examples where real-time DL is utilized to handle user speech and output smart responses. The conventional single-threaded NPUs can only store the old thread's context to memory and retrieve the new thread's context using preemption mechanisms [14] to run another task. However, it will have a large context switch penalty. In our multi-threaded NPU for LSTMs, the new task can execute from another thread as soon as it comes, as shown in Figure 4. Please note that a particular thread may still stall due to data hazard, but the physical core is not stalled, since multiple threads share the same computational physical core, which leads to "Virtual idle" as shown in this figure. We believe that further optimizations, e.g., **simultaneous multithreading (SMT)** can be adopted to our NPU design to gain even higher performance. We leave it for our future work, because it does not have a big impact on the conclusions we draw from this work.

3.2 Weights Matrix Blocking Strategy

The LSTM calculation of one timestep in an LSTM layer has four MVM operations according to the Equations (1). Besides, these four MVM are independent and share the same size. Since the four matrices of i, f, o, u gates in LSTMs have the same size, these matrices can be combined into a single large matrix [1, 57]. Figure 5 illustrates the basic idea. Thus, in the calculation of one timestep of an LSTM layer, we can only focus on the optimizations of one large matrix multiplying one vector for the whole LSTM cell rather than four small matrices multiplied by one vector. This is a general optimization that can be utilized for any MVM operations that share the same input vector. Because each matrix from LSTM gates has a size of $Lh \times (Lx + Lh)$, the large fused matrix has a size of $(4 \times Lh) \times (Lx + Lh)$.

Usually, deep neural networks have many compute intensive operations involving large weight matrices, which can be accelerated by running in parallel. However, when deploying on FPGAs, the parallelism is constrained by the limited hardware resources on the targeted FPGAs. It means that the whole MVM computation may not be fully unrolled and performed at once, especially for the large LSTMs. To use the computational resources efficiently, the combined weight matrix of an LSTM layer is partitioned into multiple sub-layers in advance depending on the configuration of the accelerator cores and LSTM layer sizes. Specifically, an LSTM layer of one timestep is partitioned into a number of sub-layers with equal size, as shown in Figure 6(a). For simplicity, the number of the sub-layers in the example is set as 2 to illustrate the idea, but a real design could have more sub-layers, and it depends on the design constraints. Then, **Element-based Parallelism**

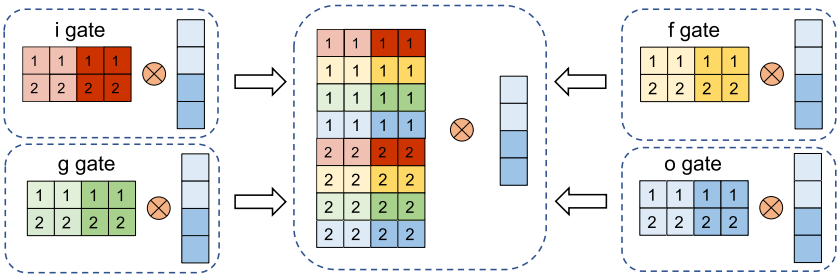

Fig. 5. The four weight matrices from the four LSTM gates and the fused weight matrix. The weights from the four gates are interleaved. The lengths of the input vector and hidden vector are 2 in this figure.

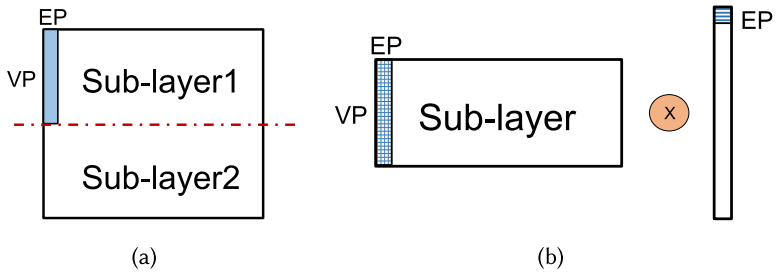

Fig. 6. The partition of an LSTM weight matrix. (a) The fused weight matrix, showing two sub-layers. (b) The MVM of a sub-layer based on a tile of $EP \times VP$ shaded in blue.

(EP) and **Vector-based Parallelism (VP)** are applied to exploit the available parallelism [57]. The number of the sub-layers is determined by $\frac{(4\times Lh)}{VP}$. The sub-layer is then divided into many small tiles, each having a size of (EP, VP), as shown in Figure 6(b). In each cycle, our Remarn core is able to process the MVM of a tile and a sub-vector of $[x_t, h_{t-1}]$ with a size of EP. To increase the design parallelism, VP should be chosen to be as large as possible. However, the largest value of VP equals Hw, which is $4 \times Lh$, since there are only four gates in an LSTM. Thus, the smallest number of sub-layers is one when VP is $4 \times Lh$. In this study, sub-layer granularity scheduling is adopted with fine-grain tasks of sub-layers for multi-LSTM execution. Besides, we interleave the rows of the four weight matrices, so that the related elements from the four gates output are adjacent in the result vector. Thus, the LSTM gate outputs can multiply with each other easily using the element-wise operations in the LSTM-tail unit. It also means that there is no data dependency between these sub-layers. The optimization of interleaving also removes the requirement to buffer large intermediate outputs from four LSTM gates, since the result sub-vectors from the sub-layers are not related and will be reduced in the tail unit soon. Each sub-vector of the MVM output can be handled individually in the LSTM-tail units. There is no need to wait for other sub-layer results to get the LSTM memory cell status and hidden vector of the current sub-layer. The proposed CGMT core will always finish processing all the sub-layers in one timestep before it switches to another thread, because the data hazard in the LSTMs happens when timestep changes. Compared with a fine-grained multithreading scheme that switches the context between threads in every cycle, we can avoid buffering these large intermediate MVM operation values as well as element-wise operation values, since these values will finally form the sub-vectors of LSTM memory cells and hidden units. Only the thread dedicated buffers are needed to be added for storing the LSTM memory cells and hidden units in each timestep, since different threads process different LSTM tasks.

3.3 Multi-core for Recurrent Neural Processing Unit

Many RNN/LSTM accelerators deploy all the computing resources to form a single accelerator core with a large scale, leveraging data-level parallelism. However, this type of accelerator is only able to process one task at a time, resulting in potential underutilization and inefficiency when multiple tasks come at the same time, especially in large data centers. Multithreading described in Section 3.1 can address the issue partially but it still cannot easily deal with a large number of small workloads. Recently, spatial architectures have become a popular solution to build high throughput accelerators for DNNs [24, 78, 82]. Generally, multiple PEs are grouped as an engine, and these engines are connected with an on-chip interconnection to enable data communication during processing. f-CNN^x [78] proposes multiple custom dedicated engines for the workloads with various CNN models. Planaria [24] proposes to dynamically fission a systolic array-based DNN accelerator at runtime to spatially co-locate multiple DNN inferences. However, none of them targets RNNs, and they do not consider the recurrent nature and data dependency in RNN computations that are absent from accelerators targeting CNNs and **fully connected (FC)** layers. This work adopts a similar idea to Reference [24]. But instead of using a systolic two-dimensional matrix-matrix multiplication array, our architecture uses matrix-vector multiplication engines, since RNNs and MLPs are dominated by matrix-vector multiplication operations. With multiple sub-accelerator cores, it enables spatial co-execution of multiple RNN/LSTM inferences on the same hardware and offers simultaneous multi-RNN accelerations.

In addition to inter-model parallelism, which is naturally supported by using multiple sub-accelerator cores, such as co-execution of multiple models, our design also enables intra-model parallelism among different layers and timesteps of the RNN layers. Generally, in an RNN model, a latter layer or timestep should wait until its preceding layer or timestep finishes, since some of its inputs are the output of the preceding layer or timestep, as shown in Figure 7(a). For example, the computation that requires c_{00} and h_{00} cannot start until they are available from the preceding layer or timestep. Using a single-threaded single-core accelerator, two LSTM workloads will run as Figure 7(c) shows. The accelerator will process the LSTM layers and timesteps iteratively. The temporal inefficiency comes from the stall of data dependencies and the spatial inefficiency comes from mapping a small network to a large core leaving some resources idle. With a multi-threaded accelerator, each layer of the LSTM can run on different threads to remove the stall or the temporal inefficiency that comes from the RNN nature data dependencies. Please note, a special computation order, as shown in Figure 7(b) is required to achieve stall-free processing using a multi-threaded single-core accelerators, as shown in Figure 7(d). However, when LSTMs are small, many resources are left unused when mapping them to a large core. In such cases, a small core will get a similar latency to the one using a large core. Besides, with multiple sub-accelerator cores, the processing of the cascaded LSTM layers can be overlapped in a layerwise coarse-grained pipeline as shown in Figure 7(e). The second layer does not need to wait for the whole sequence of hidden vectors of the first layer to be ready. Just one hidden vector from the preceding LSTM layer is sufficient to start the calculation of the next LSTM layer. It helps to reduce the overall design latency. Besides, since the output of the preceding layer sinks directly in the following sub-accelerator core, there is no need for buffering large intermediate outputs, which could help to improve the area efficiency. We can further optimize the processing after combining the multithreading with multi-core. Figure 7(f) shows two LSTMs are mapping to a dual-thread three-core accelerator, which achieves the best total latency.

While the multi-core scheme leads to high performance, it also presents a new challenge: In some situations it cannot provide sufficient bus bandwidth per core. In Remarn, the multiple cores share the same front side bus. But this will not affect on the bus bandwidth, because we do not add

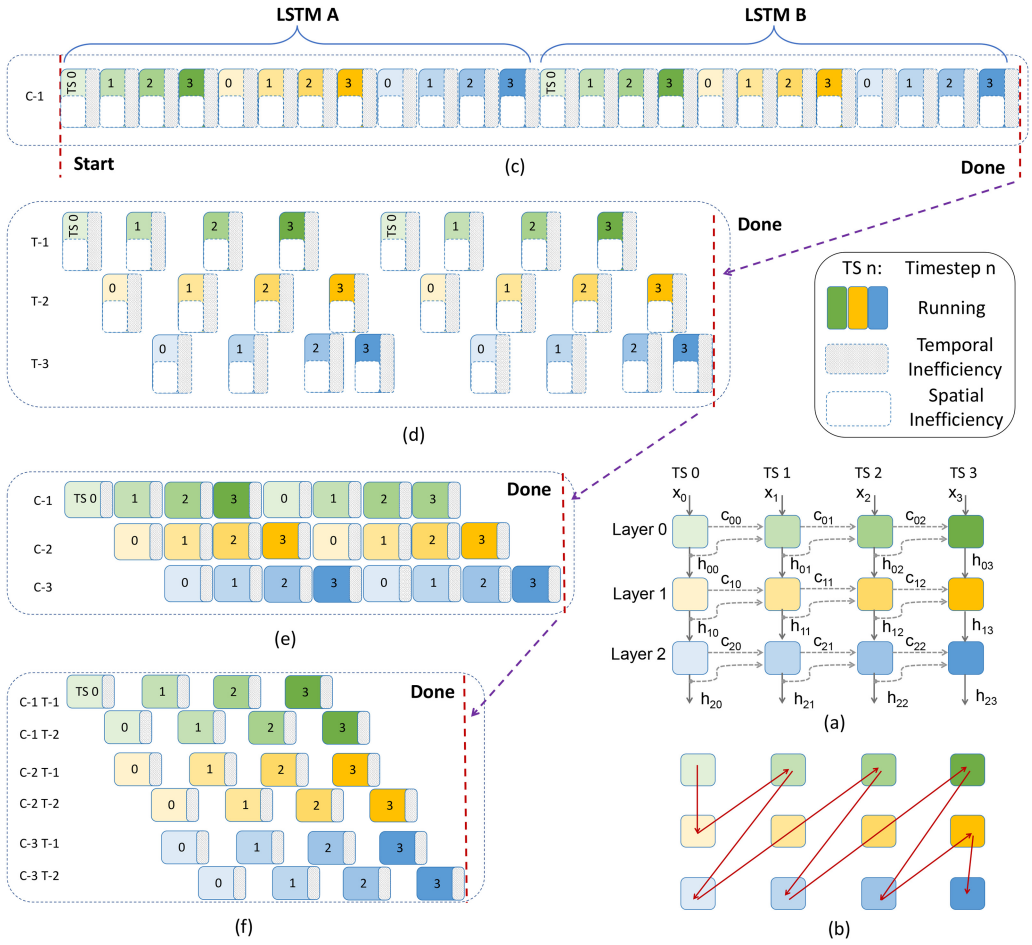

Fig. 7. Two LSTMs (A and B) are mapped to Remarn. (a) A multi-layer LSTM model showing intra-model data dependencies after unfolding. The number of layers is three and the number of timesteps is four in this example. Layer 0/1/2 are labeled in green/yellow/blue respectively. c_{ij} represents the cell state of layer i and timestep j. (b) Computation order for minimum data dependencies. (c) Two LSTMs are mapped to a single-thread single-core design. (d) Two LSTMs are mapped to a three-thread single-core design. (e) Two LSTMs are mapped to a single-thread three-core design. (f) Two LSTMs are mapped to a dual-thread three-core design.

new Remarn big cores but only split the original big core into several small ones, e.g., four small cores. When the four small cores co-operate to handle a large network, the quad-core requires the same bandwidth as the one with a big monolithic core. The total bus bandwidth they require is the same as the large core from which they are derived, which is smaller than that for four large cores. For example, the bandwidth requirement of four small LSTM models with $L_h = 64$ is much lower than the one of one big LSTM model with $L_h = 1{,}024$.

4 HARDWARE ARCHITECTURE

Based on the optimization techniques introduced above, we implement the proposed Remarn on top of a **state-of-the-art (SOTA)** RNN accelerator [57] for low-latency cloud-based applications.

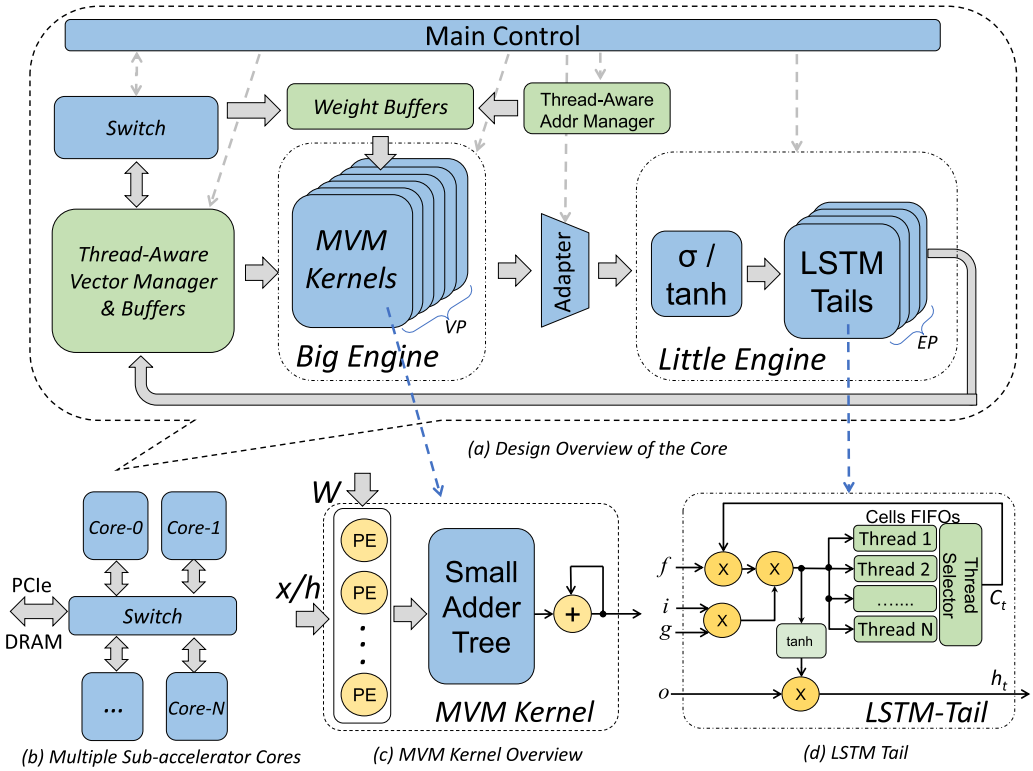

Fig. 8. Overview of the design.

In Section 4.2, we outline the main components of the architecture and detail the necessary hardware modifications required to support our Remarn unit. Several FPGA-specific optimizations are also introduced.

4.1 System Overview

A high-level block diagram of Remarn accelerator is shown in Figures 8(a) and 8(b). The original monolithic accelerator with all the MVM kernels in our previous work [59] has been split into N (e.g., $N = 4$ in the figure) sub-accelerator cores, each working in a big-little engines architecture. All these N cores are connected via the switch for hidden vectors and partial sum data movement. When N is small as in this case, the switch can be implemented by a crossbar. When N is large, then a network-on-chip could be more effective. Hence, in one extreme, all four cores can be attached together to construct a large logical accelerator, running only one RNN inference using the entire accelerator. Alternatively, it can also provide four stand-alone sub-accelerator cores, spatially co-executing 4 different RNNs simultaneously. Combining with the CGMT of four threads, it can support up to 16 different RNN inferences simultaneously. In this work, all the sub-accelerator cores are identical. But the design can be extended easily to employ a different accelerator hardware architecture, e.g., a systolic array-based one, for some of the cores to support the heterogeneous acceleration of multi-DNN workloads [40]. This work utilizes a general switch as the interconnection between the cores, which brings a slight performance drop when fusing as a large accelerator compared to a monolithic one, since the computational engines now need one more hop to share the partial results. We leave that for future work, since it has a limited impact on the conclusions we draw from our study in this article.

4.2 The Big-Little Engines Architecture

This work proposes a hardware architecture with big-little engines for the DNN accelerators, which is illustrated in Figure 8(a). The proposed accelerator core consists of a big engine, a little engine, an adapter unit, the crossbar as well as weight and vector buffers. The big engine is for the matrix operations that are computation intensive while the little one is for the small but essential components in the neural networks, such as ReLU, pooling, batch normalization, activation, and so on. The big-little engines architecture follows a wait-free single-producer/single-consumer mechanism using the adapter unit, which adjusts the data parallelism between the two engines. In this work, the big engine unit has VP MVM kernels to perform matrix-vector multiplications for LSTM gates operations. It occupies the major computational resources, such as DSPs. Practically, the design will deploy a large number of hardware resources in the big engine to increase the data parallelism to improve the processing ability. However, since the big engine produces one output vector after multiple accumulation cycles, the little engine does not require a large parallelization factor by deploying many LSTM tail units like the one in the big engine unit. The adapter unit can convert the parallelism between the two engines. With a proper adapting factor between the big engine output and little engine input, the little engine can be fully occupied without a stall or wait. The little engine unit has a vector activation function unit and EP tail units that execute the element-wise operations in the LSTM computation. It does not contain many computation resources like the big engine, but it is essential for running the whole neural networks on-chip.

Besides, with different hardware components in the little engine, such as ReLU, pooling, batch normalization, and so on, our design can easily be extended to support other types of neural networks, such as CNNs. The Brainwave [19] accelerate and serve CNN models using its MVM engines and custom multi-function units. Sometimes, to accelerate DNN designs, the big engine can be a large systolic array supporting large matrix operations and the little engine utilizes a SIMD vector unit to support general vector operations [24] for the other operations in NNs. The LSTMs are much more challenging to accelerate, exhibiting lower available parallelism and more data dependencies than two-dimensional (2D) CNNs [19]. This design focuses on accelerating RNN/LSTM inferences using a Brainwave-like architecture, but the proposed optimizations and hardware architecture can be easily extended to support other neural networks, which will be our future work.

4.3 Detailed Hardware Components in One Sub-accelerator Core

4.3.1 Overview of the Sub-accelerator Core. Each sub-accelerator core is a full-fledged RNN neural processing unit with coarse-grained multithreading. It has many logical NPUs that share nearly all resources of the physical NPU, e.g., weights buffer, kernel units, activation function units, element-wise units, as shown in Figure 8(a). The core has VP MVM kernels, each having EP PEs, resulting in $VP \times EP$ available PEs. The VP and EP values are determined via the design space exploration described in Reference [57]. In this article, each PE is one fully pipelined multiplier. The kernels are used to perform the matrix-vector multiplications between the weights and x_t as well as h_{t-1} that is required in LSTM gates operations shown in Equations (1).

4.3.2 The MVM Kernel Units. Generally, the row-wise MVM is based on an architecture with parallel multipliers followed by a balanced adder tree. Accumulating the products of these multiplications is usually achieved using a balanced adder tree structure so that a number of related additions can be scheduled in parallel, which minimizes the overall latency of the system. This architecture of the kernel unit is commonplace in FPGA-based designs of RNNs [26, 69]. Since the elements in the partial result vector are not related, we adopt the columnwise MVM [57] that is based on the architecture of parallel multipliers followed by parallel accumulators. Besides, to support element-based parallelism, a small balanced adder tree is placed between the multipliers

and the accumulators as shown in Figure 8(c). This adder tree can help to balance EP and VP to improve parallelism.

In our proposed multi-threaded NPU core, all the logical cores share all the MVM kernel units. When one logical core (thread) is stalled, the other can still conduct calculations using the same kernel units. Thus, the proposed CGMT NPU core can utilize the resources of MVM kernels efficiently to improve the NPU performance and hardware utilization. Since each thread requires its own input vector x_t and h_{t-1}, the design has to maintain multiple contexts by using buffers in the thread-aware vector manager and buffers unit. The thread selection logic is necessary to choose the required buffer to retrieve the data and conduct the remaining computations.

4.3.3 The Activation Function Unit. The σ / $tanh$ unit performs the vector-based sigmoid (σ) and hyperbolic tangent ($tanh$) operations. They are implemented using programmable lookup tables with size of 2,048 similar to References [28, 51]. The implementation using lookup tables brings several benefits. First, our NPU can run a trained model with custom activation functions (e.g., a hard sigmoid from Keras [15]) from our users without re-training of the model. Because the equations of the custom sigmoid/tanh are not changed in the model, re-training is unnecessary. More importantly, we do not touch the sensitive data of users, which is vital for many users. Second, the lookup table has a fixed latency (e.g., one cycle) but other implementations, e.g., a piecewise linear approximation, may involve multiplications that have a much larger pipe stage and latency.

4.3.4 The LSTM Tail Units. Figure 8(d) illustrates the LSTM-tail units that perform the element-wise operations in the Equations (1). The LSTM memory cell FIFOs are employed to temporarily store the status of the running LSTMs, since one thread may be switched due to data hazard before the design finish the calculation of the current LSTM layer with multiple timesteps. Because these threads are performing different LSTM layers or models, the design must keep multiple contexts of cell status of LSTMs, as shown in Figure 8(d). Other hardware contexts, such as the input data and hidden units, are maintained in the thread-aware vector manager and buffer unit with the same mechanism.

4.4 FPGA-Specific Optimizations

Since the proposed accelerator core can process a small tile with a size of (EP, VP) in each cycle, all the MVM kernel units in one core share the same input of a sub-vector of (x_t, h_{t-1}), which has EP elements. These EP elements are broadcasted to all the MVM kernels for the computation. In addition, the design also needs to broadcast a single address to all the weight buffers units to fetch the weights. To alleviate the issue of large fan-out, the tree-shaped [51] inter-connection is adopted to decrease the fan-out of each node with pipeline stages between the source and destination. The RTL code is carefully written to enable the use of HyperFlex registers on Stratix 10 to get high operating frequency [32, 76].

The DSP blocks in modern FPGAs, which are highly configurable, are often underutilized when implementing 8-bit DNN systems. [84] illustrates methods to extract two INT8 multipliers from Xilinx DSP48E2 Blocks that contain a 27×18 multiplier. Reference [41] introduces a method to pack 2 INT8 multiplications into one INT18 multiplier with extra ALMs. Both the methods require two multiplications to share one input operand. In the columnwise architecture [57], the computation order of MVM is different from the one in row-wise MVM. With the columnwise MVM used in RNN designs, one column of the weights matrix naturally shares the same element of the input vector and conducts the multiplications at the same time. Thus, these multiplications share one input operand, which helps us to pack four INT8 multiplications into one DSP block on Intel FPGAs [41] to decrease the hardware resources. In addition, this would not be a restriction (and will come

Table 2. Benchmarks of Various LSTMs for This Study

Name	Length (h_t)	Length (x_t)	Domain
Telemanom [31]	64 / 128	25	Anomaly Detection
IMDB [45]	128	128	Sentiment Classification
LRCN [17]	256	2048	Activity Recognition
Show & Tell [79]	512	512	Image Caption Generation
DeepBench [29]	256 / 512 / 1024	256 / 512 / 1024	Speech Recognition

Table 3. Resource Utilization

	Thread	Core	ALMs	M20K	DSP	Freq.
Stratix 10 (2800)	Single	Single	487,232 (52.2%)	10,061 (85.8%)	4,368 (76%)	260 MHz
Stratix 10 (2800)	Dual	Quad	533,722 (57.2%)	10,157 (86.6%)	4,368 (76%)	260 MHz

at a lower cost) if we apply a novel DSP similar to the one that was presented in Reference [7] and will be adopted in the next generation Agilex devices [33].

5 EXPERIMENTAL RESULTS AND ANALYSIS

This section introduces the experimental results on an Intel Stratix 10 FPGA that show the advances of the Remarn for RNN/LSTM accelerations.

5.1 Experimental Setup

Table 2 lists some benchmark workloads that are chosen from several typical LSTM applications to make a fair and direct comparison of our design with others. Multi-tasked LSTM workloads are constructed randomly from these LSTM workloads. We evaluate the Remarn on an Intel Stratix 10 2800 (S10) and compare the results with other work. The Remarn runs the inferences of persistent LSTM models that store the weights in on-chip memory [19, 51, 57, 59, 66–68]. Remarn is designed using Verilog RTL. And Quartus Prime Pro 19.4 is used for the compilation of FPGA designs.

The choice of the number of sub-core is based on the size of the benchmark LSTM workloads and the accelerator hardware architecture. The value of EP, explored in Reference [57], is set to 16, the same size as the previous designs [57, 59]. The value of VP should be chosen to be as large as possible. However, the largest effective value of VP is $4L_h$ as discussed in Section 3.2. Hence, the $VP = 256$ in this work, since the $L_h = 64$ for the smallest LSTM benchmark as shown in Table 2, resulting in $256 \times 16 = 4,096$ effective PEs for each sub-accelerator core. To make a fair comparison, the number of sub-accelerator cores, N, is set to 4 so that Remarn will have 16,386 PEs, which is the same as the number of total PEs in our previous designs [57, 59].

5.2 Resource Utilization

Table 3 shows the resource utilization of our designs with two configurations on FPGAs. The first design is a baseline design that is a single-threaded design without partition. It utilizes the parameter of (EP, VP) as (16, 1024), which includes 16,384 8-bit multipliers in the MVM kernels implemented using DSP blocks with extra ALMs. The second design is a dual-threaded quad-core Remarn that has four sub-accelerator cores. Each core utilizes the parameter of (EP, VP) as (16, 256). Thus, it has the same number of multipliers as the baseline design. The dual-threaded quad-core design consumes 5.0% more of total logic (ALMs) resources and 0.8% more block ram (M20K) than the baseline design. Since dual threads share the same physical core it consumes the same DSP resources as the single-threaded one.

Table 4. Performance Comparison of the Remarn versus CPUs and GPUs

	CPU	GPU	Remarn[a]
Platform	Intel Xeon Skylake	Tesla V100	Stratix 10 2800
Frequency	2.0 GHz	1.38 GHz	260 MHz
Technology	14 nm	12 nm	14 nm
Precision	F32	F16	INT8
LSTM Size (x_t, h_t)	(1024, 1024)		
TDP Power (W)	15	300	125[b]
Throughput (GOPS)	8	1180	7670
Power Effi. (GOPS/W)	0.53	3.93	61.36

[a]Dual-threaded Quad-core mode.
[b]TDP Power.

5.3 Performance and Efficiency Comparison with CPUs and GPUs

To compare the performance of the proposed design on FPGA with other platforms, the DeepBench published results [89] on an Intel Xeon Syklake CPU and NVIDIA Tesla V100 GPU are used for comparison. TensorFlow is used for both CPU and GPU. Besides, the CPU with AVX2 Vector instructions enabled is utilized while the CuDNN libraries are enabled for the GPU. cuDNN is a GPU-accelerator Library from NVIDIA, which is specialized for deep learning. Both CPU and GPU implementations run with a batch size of 1, which provides the lowest service latency of cloud, since requests are required to be processed as soon as they arrive. For a fair comparison with the throughput of our dual-threaded Remarn, the throughput of the CPU and GPU have been doubled in Table 4. The GPU is significantly underutilized even when cuDNN library API calls, since it is designed for throughput-oriented workloads. It prefers BLAS level-3 (matrix-matrix) operations that are not common in RNN computations [89]. Our FPGA design of dual-threaded Remarn achieves 6.5 times higher performance and 15.6 times higher power efficiency, respectively, than the one running on the Tesla V100 GPU as shown in Table 4.

5.4 Performance and Efficiency Comparison with the Baseline

To show the benefits of the proposed approaches, we compare the multi-threaded multi-core Remarn with the baseline design [57] in Figures 9 and 10. The baseline is a single-threaded single-core accelerator. Hardware utilization is the percentage of the peak **Tera Operations Per Second (TOPS)** achieved for each LSTM, as compared with the peak TOPS of the design with all the PEs. Figure 9 shows the speedup of the quad-core Remarn over the baseline design [57]. With the proposed approaches of CGMT and multi-core, the quad-core Remarn designs of a single thread (1-T-Q), dual threads (2-T-Q) and quad threads (4-T-Q) achieve 1.82, 2.91, and 3.16 times higher performance respectively than the baseline when targeting mixed requests from LSTM workloads with $h = 64, 128$, and 256. Please note that Remarn has the same number of PEs as the baseline, and they consume the same DSP blocks on FPGAs. Particularly, with four threads (4-T-Q), Remarn achieves 14.44 times higher performance than the baseline [57] when targeting the small LSTMs ($L_h = 64$).

When only targeting large LSTMs, the performance gain of the quad-core Remarn is small, since the baseline design [57] already achieves high hardware utilization for these LSTMs. However, the baseline still suffers from low utilization when targeting small-sized LSTMs that are commonly used in many applications [17, 55]. LSTM models that have a large number of timesteps but use small sizes are the most tangible examples that require dealing with lots of dependencies, as well as the parallel task of MVMs [86]. Our proposed approach and hardware architecture can alleviate

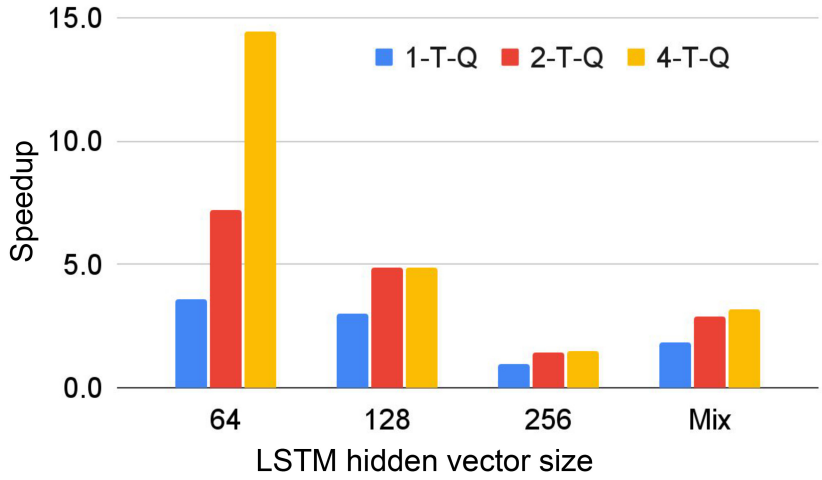

Fig. 9. Performance speedup for quad-core Remarn with various threads.

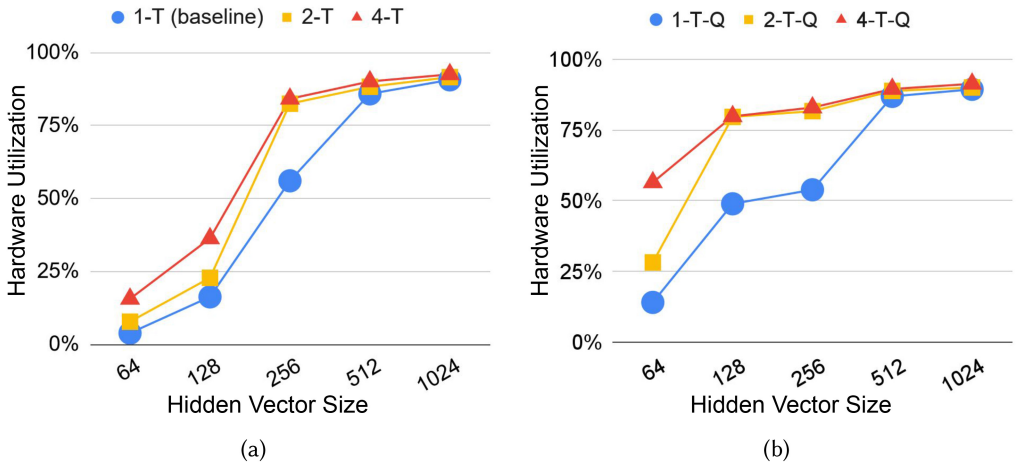

Fig. 10. Hardware utilization targeting various LSTM tasks. (a) Hardware utilization of different LSTM tasks using various threads with a single big accelerator. (b) Hardware utilization of different LSTM tasks using various threads with four sub-accelerator cores.

this problem by leveraging the coarse-grained multithreading combining the spatial co-execution using multiple sub-accelerator cores. With the demand for high-performance designs, it is vital to maximize the hardware utilization to attain the best possible effective performance and power efficiency.

When compared to our previous work [59] with only multithreading, Remarn can attain much higher hardware utilization and performance. Remarn achieves higher than 50% utilization for all the LSTM cases with $L_h = 128$ as shown in Figure 10(b) while the previous work can only achieve up to 36.4%. Particularly, with dual threads, the previous work achieves the utilization lower than 25% for LSTMs with $L_h = 128$; however, the proposed Remarn can achieve higher than 75%. Compared to our previous work [59], the Remarn proposed in this work achieves up to 3.61 times higher performance when targeting the small LSTMs, as shown in Figure 11, which

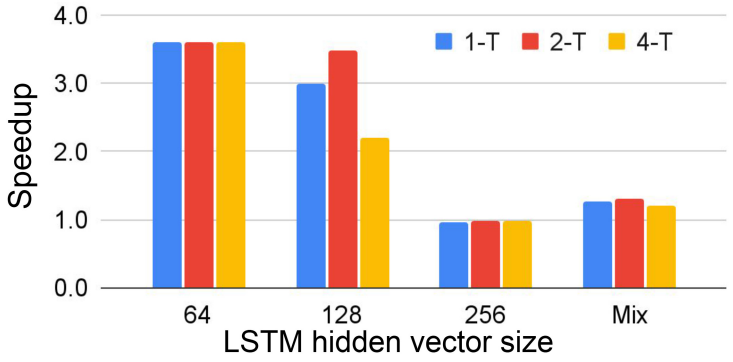

Fig. 11. Performance speedup of the multi-threaded quad-core Remarn over the multi-threaded single-core design.

shows the advances of the proposed architecture using multiple sub-accelerator cores. There is a small reduction from a linear speedup for the model size of 64, which is due to control logic added to each small cores when splitting the big engine. The impact is negligible when the workload is large with a long runtime. We believe this can be optimized with more efforts but doing so has little impact on the conclusions we draw from this study.

The design with multithreading alone shows a steady increase from 1-T to 2-T and to 4-T [59] for the model size of 128. However, the additional speedups for the quad-core design dropped when it is from 2-T to 4-T threads as shown in Figure 11. It is because with 2-T on small cores, the design can already remove most of the stalls caused by the RNN data dependencies for the model size of 128. Hence, the quad-core design with 2-T and 4-T achieves a similar performance and runtime hardware utilization as shown in Figure 10(b). But with 2-T on the big core, the design still suffers from some stalls so the performance of 4-T on the big core is much better than 2-T. Hence, the additional speedups drop when going from 2-T to 4-T threads on the quad-core design.

Moreover, when compared to References [57, 59], which has a single large core, the quad-core Remarn has a slight performance drop (less than 2%) when all 4 cores are working together as a big accelerator to run a large workload, as shown in Tables 4 and 5. It is because the multi-core design can lead to synchronization costs than the one using a monolithic accelerator. When four cores are running the same large workload, a synchronization mechanism is required to maintain the correct data dependencies in RNNs. It is a design tradeoff. With slight performance loss on the large workloads, the design can achieve much higher performance on the small workloads, which can result in a much better performance gain for all the workloads. Besides, the users can re-configure the FPGA into a big and monolithic accelerator when most of large RNN workloads are large, which can achieve the best performance.

The evaluation results also show that Remarn gets a low utilization when running small LSTMs ($h = 64$), as shown in Figure 10(b). Even with four threads, the design still stalls, because the cycles of processing MVMs in the other threads still cannot completely hide the whole pipeline latency to get the h_t available before it is required for the next timestep. The number of cycles to process the MVM of the LSTM gates is given by $\frac{Lx+Lh}{EP}$. Different choices of (EP, VP) impact the performance and utilization of the designs that run LSTMs with different sizes. There is a tradeoff between the extra performance that can be achieved and the hardware design complexity as well as extra hardware resources of supporting various sets of (EP, VP) at runtime. It is our future work to enhance this accelerator architecture to support various sets of (EP, VP), since different sizes of LSTMs may prefer different optimal EP and VP parameters.

Table 5. Comparison with Existing Implementations of LSTMs on Stratix 10 GX 2800

	ISCA18-BW [19]	FCCM19-NPU [51]	FCCM20-NPU [57]	Remarn[a]
FPGA	Stratix 10 GX2800			
Model Storage	On-chip			
Precision (bits)	BFP-1s5e2m	8 fixed	8 fixed	8 fixed
DSP Used	5245 (91%)	4880 (85%)	4368 (76%)	4368 (76%)
Frequency (MHz)	250	260	260	260
Power[b] (W)	125			
LSTM Size (h_t)	256			
Performance (GOPS)	370	1431	4790	6965
Power[b] Effi. (GOPS/W)	2.96	11.45	38.32	55.72
LSTM HW Utilization	0.8%	14.3%	56.1%	81.6%

[a]Dual-threaded Quad-core mode.
[b]TDP Power is used.

5.5 Performance and Efficiency Comparison with Other FPGA-based Designs

Table 5 shows the comparison between our Remarn and the existing LSTM accelerator implementations using a Stratix 10 GX 2800 FPGA. For a fair comparison, we only show the previous work with a detailed implementation of the LSTM system using the same FPGA. We list the model storage, precision, DSP used, runtime frequency, power, throughput and power efficiency as well as the **hardware (HW)** utilization. The **thermal design power (TDP)** that is 125 W is utilized for a fair comparison, since it is reported in all the listed work. Overall, Remarn provides over 4.87 times higher performance and 5.71 times higher hardware utilization than the SOTA design [51] when targeting LSTM models with h_t=256, as shown in Table 5. Compared with the baseline that is a single-threaded single-core design [57], Remarn achieves 1.45 to 14.4 higher throughput and HW utilization. Since multithreading also ensures there is no downside to peak performance in a single-threaded mode, the Remarn can achieve a similar peak performance with the one in Reference [57], which is the highest with respect to the SOTA FPGA-based RNN/LSTM designs with a commonly used INT8 precision. The only prior work that has a higher peak performance is Reference [19] that employs an 8-bit block floating-point representation. However, when targeting the small LSTM models, the throughput of the proposed Remarn is 18.82 times higher than [19] as shown in Table 5. Moreover, Remarn achieves the highest hardware utilization among all the listed designs across various LSTM models. This work focuses on maximizing throughput and minimizing latency by increasing HW utilization.

5.6 Scheduling

The scheduling strategy for multi-core accelerators for DNN workloads is a new and rarely explored topic. PREMA [14] introduces a preemptive scheduling algorithm on a single-core NPU to support multi-DNN execution. AI-MultiTasking [4] proposes to process different neural networks by partitioning their various layers into identical sub-layers in compile time. It also develops a heuristic of mapping DNN jobs to the accelerator. In Reference [40], **Heterogeneous Dataflow accelerators (HDAs)** are proposed to support the heterogeneous acceleration of multi-DNN workloads using manual-designed mapping of jobs to sub-accelerators and compile time layerwise scheduling. LayerWeaver [53] introduces a layerwise time-multiplexing scheduler on single NPU by interweaving layer execution to increase the hardware utilization. Besides, MAGMA [36] proposes a novel framework for mapping multiple DNNs on multiple accelerator cores.

The focus of this article is on the accelerator architecture, and on examples of optimal utilization of that architecture based on fine-grained scheduling of tasks made up of LSTM sub-layers running on sub-cores. We exploit the intra-model data dependencies of both timesteps and layers in LSTMs, as shown in Figure 7(a). Both time step and layer dependencies have a linear dependence chain in the model. For LSTM inference, it is processed from the first layer to the last layer sequentially and from the first timestep to the last timestep in each layer. Besides, layers from different LSTMs are independent. Inspired by References [4, 40], which rely on manually designed heuristics, this work develops a set of heuristics that exploit the characteristics of LSTM inference workloads to reduce the scheduling overhead for the proposed homogeneous multi-core accelerator. We do not include algorithmic details (e.g., flowchart or pseudocode) about scheduling of these sub-layers tasks, other than mentioning related research on scheduling strategies [4, 14, 40] and LayerWeaver [53] as well as the custom mapping algorithm MAGMA [36]. Architectural tradeoffs of these scheduling strategies, based on queuing theory and other techniques, are important and would be best addressed by future work. Besides, further research will cover fine-grained dynamic scheduling for our accelerator, such as using Gang scheduling that schedules a gang of jobs onto free cores. Furthermore, other sophisticated scheduling strategies relevant to our approach will be studied and tested.

5.7 Power Consumption and Optimizations

The proposed extension of the multi-core feature is based on splitting a large monolithic accelerator into several sub-accelerator cores, which does not change the total number of existing PEs in hardware. There is only a small bump in the hardware resources for the extended accelerator because of supporting new features, which may cause a little more power/energy consumption. However, the extra power consumption caused by extra FPGA hardware resources is small, since the new design only has 5% more area than the baseline [57] but can get 1.45 to 14.4 times higher performance, resulting in high power efficiency.

Besides, the architectural extensions lead to higher runtime dynamic power, since these extensions provide more parallelism and higher hardware utilization. In general, the performance and power are positively correlated; the more accelerators used, the higher the power consumption. However, higher utilization also means that our architecture wastes less power on the underutilized execution resources than the baseline [57], which increases the overall power/energy efficiency. The overall increase in utilization also means the proposed design completes the same work in a shorter period of time, resulting in a potential lower energy consumption. Besides, with multiple sub-accelerator cores, the design has the potential to totally turn off some of the cores when there is only a small amount of workload to save power, which is not possible in the baseline where only a big and monolithic core is used. In future work, we will build an architecture-level power model targeting multi-threaded multi-core DNN accelerators so that we can quantify the impact of these architectural extensions on power/energy consumption.

There are some potential optimizations for the power consumption targeting our architecture. Since the proposed design achieves a high hardware utilization, it may not require so many PEs to reach a design performance goal. Reducing the number of total PEs or reducing the hardware resources in design time can help to reduce the total power consumption. However, the design causes large dynamic power consumption because of the high utilization. Hence, one possible optimization is to control dynamic power consumption by gracefully degrading the core activities when a power consumption threshold is exceeded. It can be done by a feedback of the current power consumption of the cores, which may be estimated from some performance counters or read from some on-die sensors.

6 RELATED WORK

There are many previous studies about FPGA-based accelerations of persistent RNN/LSTM with weights stored in on-chip memory [19, 51, 58, 66–69]. Rybalkin et al. [69] introduces the hardware architecture of BiLSTM targeting OCR applications. All the weights and activations are quantized to 5-bit and stored in on-chip memory. [66] quantizes the design using 1-8 bits and achieves an accuracy that surpasses the one using floating-point on a given dataset. Besides, their later work [67, 68] proposes a novel hardware architecture of 2D-LSTM and investigates the tradeoff between precision and performance. There are also many previous studies about LSTM implementations with weights stored in off-chip memory on FPGAs [10, 26, 27, 54, 61], which has been recognized as the performance bottleneck. In LSTM computations, a weights matrix tends to be repeatedly loaded from off-chip memory if the size of on-chip memory is not large enough to store the entire matrix, which brings large latency. Guan et al. [27] proposes a smart memory organization with on-chip double buffers to overlap computations with data transfers. And References [22, 72] apply the batching technique to increase the throughput of LSTM inferences. For example, E-BATCH is proposed [72] for RNNs, which increases throughput while also improving energy efficiency on an ASIC-based accelerator. Apart from batching, References [54, 61, 63] introduce novel LSTM weights reuse schemes that utilizes the weights sharing characteristics in different timestep computations in one inference. These schemes reduce the access of the off-chip memory and decrease the energy cost as well as improve the design throughput.

Some of the previous work [42, 80, 81, 85] adopts circulant matrix to optimize LSTMs by reducing the weight matrices of LSTM inferences. Besides, an approximate computing scheme is deployed for LSTMs using small tiles [65]. And stochastic computing is used to improve the energy efficiency of the RNN inference [43]. Reference [37] proposes an LSTM architecture based on reversible logic gates for low power circuit designs. POLAR [5] and BRDS [23] present FPGA-based pipelined and overlapped architecture for dense and sparse LSTM inferences, respectively. A multi-FPGA approach [74] is proposed to accelerate multi-layer RNNs. It achieves a single-layer latency targeting deep RNNs with arbitrarily multiple layers using an FPGA-based cluster. Reference [49] presents a multi-FPGA-based architecture to accelerate neural machine translation. Reference [39] explores various partitioning strategies of large RNN inferences to achieve scalable multi-FPGA acceleration. It also analyses the performance impact of software pipelining and collective communications. PDL-FGPU [44], a specialized FPGA-based GPU overlay architecture is proposed to run persistent Deep Learning, including various RNN variants. Initiation interval (II) balancing [62] is proposed with a layerwise implementation of LSTMs to achieve ultra low latency. The layerwise implementation of RNNs is also used in accelerating Bayesian RNNs [18].

There is also much previous work [9, 11, 13, 20, 21, 28, 34, 48, 70, 83, 90] exploit the sparsity of data and weights with pruning to reduce the NN computation and also the memory footprint to achieve high performance and efficiencies. ESE [28] proposes a pruning technique that compresses a large LSTM model by 20× without sacrificing the prediction accuracy. DeltaRNN [21] utilizes the Delta Network algorithm to reduce MxV operations and corresponding weight fetches by skipping dispensable computations during inference of RNNs. It updates the output of a neuron only when the neuron's activation changes by more than a delta threshold. Bank-Balanced Sparsity [9] is proposed to achieve both high prediction accuracy of a pruned LSTM model and high hardware efficiency of the model running on FPGAs. It partitions weight matrix rows into banks for parallel computing and adopts fine-grained pruning inside each bank to maintain model accuracy. BLINK [11] designs the LSTM inference using a bit-sparse data representation. And it turns multiplications into bit-shift operation to improve the energy efficiency while maintaining the LSTM inference accuracy for real-time calcium image processing. The extension [12] proposes a

combination of the bit-sparse quantization and the pruning methods for energy-efficient LSTM inferences. More recently, Spartus [20] exploits spatio-temporal sparsity to achieve ultra-low-latency inference using Column-Balanced Targeted Dropout. These studies are orthogonal to our proposed approach and hardware architecture. These techniques can be complementary to our approaches to achieve even higher performance and efficiency of RNN/LSTM inferences using FPGAs.

The Brainwave design [19] is a single-threaded SIMD architecture for running real-time AI, including persistent RNNs. It achieves more than an order of magnitude improvement in latency and throughput over state-of-the-art GPUs on large memory intensive RNN models at a batch size of 1. It stores NN model weights on-chip for RNNs to get a necessary high memory read bandwidth to achieve higher performances. A **coarse-grained reconfigurable architecture (CGRA)**-based RNN accelerator is proposed [89] on top of Plasticine [56] with a set of techniques for performing cross-kernel optimization in RNN cells. AERO [38] is a single-threaded instruction-set-based processor using a versatile vector-processing unit customized for RNN inferences on resource-limited FPGAs. A Brainwave-like NPU is proposed in Reference [51] with a single-threaded architecture. They also explore the potential of combining a TensorRAM with FPGAs to provide large high-speed memory for large memory intensive RNN sequence models. Besides, their late work [8] deploys the Brainwave-like NPU on the Stratix 10 NX that is Intel's new AI-optimized FPGA featured with AI tensor blocks. [57, 60] proposes a novel latency-hiding hardware architecture based on columnwise MVM and fine-grained tiling strategy to eliminate data dependency of RNNs, which improves the design throughput of RNNs/LSTMs. However, all these NPUs targeting RNNs/LSTMs are single threaded.

There is a large demand for architectural support of multi-DNN execution to maximize hardware utilization and reduce the costs of a large-scale production system. For example, TensorRT from Nvidia supports concurrent DNN execution for users to run multi-DNN workloads on the same GPU simultaneously. SMT-SA [71] presents a simultaneous multithreading approach for systolic arrays to solve the issue of underutilization because of zero-valued inputs. However, it is not able to deal with the underutilization issue that comes from the data dependencies in RNNs/LSTMs. A multi-threaded CGRA [3] has been proposed, but it is only for CNN accelerations. Reference [50] presents to generate in-order multi-threaded processing pipelined datapath automatically with the high-level specification of an unpipelined datapath. CUSTARD [16] presents a customisable multi-threaded FPGA soft processor with features including design space exploration and a compiler for automatic selection of custom instructions. More recently, PREMA [14] proposes a preemptible NPU with a preemptive scheduling algorithm to support multi-DNN execution. However, it does not consider the data dependencies between RNN/LSTM timesteps. AI-MultiTasking [4] proposes to balance memory-intensive and compute-intensive tasks from different neural networks and process them in parallel by partitioning various layers into lots of identical sub-layers. The RNNs are handled just as FC layers in the scheduling scheme. However, RNNs are more complex than FC layers that have no data dependencies. In Reference [55], a dual-core accelerator for LSTM-RNN is proposed to execute multiple jobs simultaneously or have cores collaborate on a single job. However, they perform multithreading by utilizing the dual cores with one thread on one core, which is not an area / cost efficient way to add extra threads. Besides, there is some previous work targeting spatial multi-tenant execution. f-CNN^x [78] employs multiple custom dedicated engines for the workloads with various CNN models. Reference [64] maps multiple LSTM models to an FPGA device by introducing a framework that alters their computational structure, opening opportunities for co-optimizing the memory requirements to the target architecture via applying compression schemes across multiple LSTM models. In Reference [40] HDAs are proposed to support the heterogeneous acceleration of multi-DNN workloads. More recently, Planaria [24] introduces the dynamical fission of a systolic-array-based DNN accelerator

at runtime to support spatial co-execution of multiple DNN inferences. However, none of them targets RNNs, and they do not consider the recurrent nature and data dependency in RNN computations. This work designs a multi-threaded accelerator with fission to enable spatial and temporal co-execution of multiple RNN/LSTM inferences on the same hardware and offers simultaneous multi-RNN accelerations. It can handle multiple requests using multi-threaded mode with multiple sub-accelerator cores while still being able to run a task in a single-threaded mode with a large accelerator to attain higher performance and lower latency when targeting the workloads with high priority.

7 CONCLUSIONS AND FUTURE WORK

This work proposes Remarn, a multi-threaded multi-core NPU supporting spatial and temporal co-execution of RNN/LSTM inferences to improve the processing abilities of cloud-based NPUs. The Remarn can increase the throughput while improving the hardware utilization of the cloud-based NPUs. We have implemented the proposed multi-threaded multi-core accelerator architecture on Intel Stratix 10 FPGAs with superior performance and efficiency, which demonstrates the effectiveness of our approaches. Our study shows that both multithreading and multi sub-accelerator core techniques can address the underutilization issue (resource inefficiency) when targeting small sized RNN workloads. Besides, it also shows that multi sub-accelerator core is slightly better than multithreading for small workloads. The design of quad-core with single-thread (1-T-Q) achieves a better utilization than the design of single-core quad-thread (4-T) for the LSTM of $L_h = 128$, as shown in Figure 10. But this figure also shows that the single-core quad-thread (4-T) design gets a better performance for the LSTM of $L_h = 256$. In addition, both techniques do not bring benefits to large LSTM workloads, since the baseline design has achieved high utilization. Moreover, this study shows that multithreading targets temporal co-execution, while multi-core targets spatial co-execution, and the two techniques can be combined to achieve a much better performance. Further research includes exploring how Remarn can benefit from other enhancements such as dynamic scheduling and runtime power analysing, studying the tradeoffs of varying the number and the heterogeneity of Remarn cores, and automating the proposed approaches to enable rapid development of efficient Remarn designs for datacentres as well as edge processing and embedded systems.

REFERENCES

[1] Mohamed S. Abdelfattah, David Han, Andrew Bitar, Roberto DiCecco, Shane O'Connell, Nitika Shanker, Joseph Chu, Ian Prins, Joshua Fender, Andrew C. Ling, et al. 2018. DLA: Compiler and FPGA overlay for neural network inference acceleration. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL'18)*. IEEE, 411–4117.

[2] Dario Amodei et al. 2016. Deep speech 2: End-to-end speech recognition in english and mandarin. In *Proceedings of the International Conference on Machine Learning*.

[3] Kota Ando, Shinya Takamaeda-Yamazaki, Masayuki Ikebe, Tetsuya Asai, and Masato Motomura. 2017. A multi-threaded CGRA for convolutional neural network processing. *Circ. Syst.* 8, 6 (2017), 149–170.

[4] Eunjin Baek, Dongup Kwon, and Jangwoo Kim. 2020. A multi-neural network acceleration architecture. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*. IEEE, 940–953.

[5] Erfan Bank-Tavakoli, Seyed Abolfazl Ghasemzadeh, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2019. Polar: A pipelined/overlapped fpga-based lstm accelerator. *IEEE Trans. VLSI Syst.* 28, 3 (2019), 838–842.

[6] John M. Borkenhagen et al. 2000. A multithreaded PowerPC processor for commercial servers. *IBM J. Res. Dev.* 44, 6 (2000), 885–898.

[7] Andrew Boutros et al. 2018. Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'18)*. IEEE.

[8] Andrew Boutros, Eriko Nurvitadhi, Rui Ma, Sergey Gribok, Zhipeng Zhao, James C. Hoe, Vaughn Betz, and Martin Langhammer. 2020. Beyond peak performance: Comparing the real performance of AI-optimized FPGAs and GPUs. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'20)*. IEEE, 10–19.

[9] Shijie Cao, Chen Zhang, Zhuliang Yao, Wencong Xiao, Lanshun Nie, Dechen Zhan, Yunxin Liu, Ming Wu, and Lintao Zhang. 2019. Efficient and effective sparse LSTM on FPGA with bank-balanced sparsity. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 63–72.

[10] Andre Xian Ming Chang, Berin Martini, and Eugenio Culurciello. 2015. Recurrent neural networks hardware implementation on FPGA. arXiv:1511.05552. Retrieved from https://arxiv.org/abs/1511.05552.

[11] Zhe Chen, Garrett J. Blair, Hugh T. Blair, and Jason Cong. 2020. BLINK: Bit-sparse LSTM inference kernel enabling efficient calcium trace extraction for neurofeedback devices. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 217–222.

[12] Zhe Chen, Hugh T. Blair, and Jason Cong. 2022. Energy efficient LSTM inference accelerator for real-time causal prediction. *ACM Trans. Des. Autom. Electr. Syst.* 27, 5, Article 44 (September 2022), 19 pages. https://doi.org/10.1145/349500

[13] Zhe Chen, Andrew Howe, Hugh T. Blair, and Jason Cong. 2018. CLINK: Compact LSTM inference kernel for energy efficient neurofeedback devices. In *Proceedings of the International Symposium on Low Power Electronics and Design*. 1–6.

[14] Yujeong Choi and Minsoo Rhu. 2020. PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'20)*. IEEE, 220–233.

[15] François Chollet et al. 2015. Keras: Deep Learning Library for theano and tensorflow. https://keras.io/k.

[16] R. Dimond, O. Mencer, and W. Luk. 2006. Application-specific customisation of multi-threaded soft processors. *IEE Proc. Comput. Digit. Techn.* 153, 3 (2006), 173–180.

[17] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. 2015. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2625–2634.

[18] Martin Ferianc, Zhiqiang Que, Hongxiang Fan, Wayne Luk, and Miguel Rodrigues. 2021. Optimizing Bayesian recurrent neural networks on an FPGA-based accelerator. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'21)*. IEEE, 1–10.

[19] Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Logan Adams, Mahdi Ghandi, et al. 2018. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 1–14.

[20] Chang Gao, Tobi Delbruck, and Shih-Chii Liu. 2021. Spartus: A 9.4 TOp/s FPGA-based LSTM accelerator exploiting spatio-temporal sparsity. *IEEE Transactions on Neural Networks and Learning Systems*. (Early Access)

[21] Chang Gao, Daniel Neil, Enea Ceolini, Shih-Chii Liu, and Tobi Delbruck. 2018. DeltaRNN: A power-efficient recurrent neural network accelerator. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 21–30.

[22] Pin Gao, Lingfan Yu, Yongwei Wu, and Jinyang Li. 2018. Low latency RNN inference with cellular batching. In *Proceedings of the 13th European Conference on Computer Systems Conference (EuroSys'18)*. 1–15.

[23] Seyed Abolfazl Ghasemzadeh, Erfan Bank Tavakoli, Mehdi Kamal, Ali Afzali-Kusha, and Massoud Pedram. 2021. BRDS: An FPGA-based LSTM accelerator with row-balanced dual-ratio sparsification. arXiv:2101.02667. Retrieved from https://arxiv.org/abs/2101.02667.

[24] Soroush Ghodrati, Byung Hoon Ahn, Joon Kyung Kim, Sean Kinzer, Brahmendra Reddy Yatham, Navateja Alla, Hardik Sharma, Mohammad Alian, Eiman Ebrahimi, Nam Sung Kim, et al. 2020. Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks. In *Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. IEEE, 681–697.

[25] Yoav Goldberg. 2016. A primer on neural network models for natural language processing. *J. Artif. Intell. Res.* 57 (2016), 345–420.

[26] Yijin Guan, Hao Liang, Ningyi Xu, Wenqiang Wang, Shaoshuai Shi, Xi Chen, Guangyu Sun, Wei Zhang, and Jason Cong. 2017. FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates. In *Proceedings of the IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'17)*. IEEE, 152–159.

[27] Yijin Guan, Zhihang Yuan, Guangyu Sun, and Jason Cong. 2017. FPGA-based accelerator for long short-term memory recurrent neural networks. In *Proceedings of the 22nd Asia and South Pacific Design Automation Conference (ASP-DAC'17)*. IEEE, 629–634.

[28] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. 2017. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 75–84.

[29] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, et al. 2014. Deep speech: Scaling up end-to-end speech recognition. arXiv:1412.5567. Retrieved from https://arxiv.org/abs/1412.5567.

[30] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.* 9, 8 (1997), 1735–1780.
[31] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. 2018. Detecting spacecraft anomalies using LSTM and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 387–395.
[32] Intel. [n.d.]. *Understanding How Hyperflex Architecture Enables High Performance Systems*. White Paper 01231.
[33] Intel. 2020. *Intel Agilex Variable Precision DSP Blocks User Guide*.
[34] Jingfei Jiang, Tao Xiao, Jinwei Xu, Dong Wen, Lei Gao, and Yong Dou. 2022. A low-latency LSTM accelerator using balanced sparsity based on FPGA. *Microprocess. Microsyst.* 89 (2022), 104417.
[35] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*. 1–12.
[36] Sheng-Chun Kao and Tushar Krishna. 2022. MAGMA: An optimization framework for mapping multiple DNNs on multiple accelerator cores. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'22)*. IEEE.
[37] Kasem Khalil, Bappaditya Dey, Ashok Kumar, and Magdy Bayoumi. 2021. A reversible-logic based architecture for long short-term memory (LSTM) network. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'21)*. IEEE, 1–5.
[38] Jinwon Kim, Jiho Kim, and Tae-Hwan Kim. 2021. AERO: A 1.28 MOP/s/LUT reconfigurable inference processor for recurrent neural networks in a resource-limited FPGA. *Electronics* 10, 11 (2021), 1249.
[39] Dongup Kwon, Suyeon Hur, Hamin Jang, Eriko Nurvitadhi, and Jangwoo Kim. 2020. Scalable multi-FPGA acceleration for large RNNs with full parallelism levels. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC'20)*. IEEE, 1–6.
[40] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. 2021. Heterogeneous dataflow accelerators for multi-DNN workloads. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. IEEE, 71–83.
[41] Martin Langhammer, Bogdan Pasca, Gregg Baeckler, and Sergey Gribok. 2019. Extracting INT8 multipliers from INT18 multipliers. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL'19)*. IEEE.
[42] Zhe Li, Caiwen Ding, Siyue Wang, Wujie Wen, Youwei Zhuo, Chang Liu, Qinru Qiu, Wenyao Xu, Xue Lin, Xuehai Qian, et al. 2019. E-RNN: Design optimization for efficient recurrent neural networks in FPGAs. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA'19)*. IEEE, 69–80.
[43] Yidong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. 2019. An energy-efficient and noise-tolerant recurrent neural network using stochastic computing. *IEEE Trans. VLSI Syst.* 27, 9 (2019), 2213–2221.
[44] Rui Ma, Jia-Ching Hsu, Tian Tan, Eriko Nurvitadhi, David Sheffield, Rob Pelt, Martin Langhammer, Jaewoong Sim, Aravind Dasu, and Derek Chiou. 2021. Specializing FGPU for persistent deep learning. *ACM Trans. Reconfig. Technol. Syst.* 14, 2 (2021), 1–23.
[45] Andrew Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. 142–150.
[46] Cameron McNairy and Rohit Bhatia. 2005. Montecito: A dual-core, dual-thread itanium processor. *IEEE Micro* 25, 2 (2020), 10–20.
[47] Sparsh Mittal and Sumanth Umesh. 2021. A survey on hardware accelerators and optimization techniques for RNNs. *J. Syst. Arch.* 112 (2021), 101839.
[48] Guocai Nan, Chenghua Wang, Weiqiang Liu, and Fabrizio Lombardi. 2020. DC-LSTM: Deep compressed LSTM with low bit-width and structured matrices. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS'20)*. IEEE, 1–5.
[49] Eriko Nurvitadhi, Andrew Boutros, Prerna Budhkar, Ali Jafari, Dongup Kwon, David Sheffield, Abirami Prabhakaran, Karthik Gururaj, Pranavi Appana, and Mishali Naik. 2019. Scalable low-latency persistent neural machine translation on CPU server with multiple FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'19)*. IEEE, 307–310.
[50] Eriko Nurvitadhi, James C. Hoe, Shih-Lien L. Lu, and Timothy Kam. 2010. Automatic multithreaded pipeline synthesis from transactional datapath specifications. In *Proceedings of the Design Automation Conference*. IEEE, 314–319.
[51] Eriko Nurvitadhi, Dongup Kwon, Ali Jafari, Andrew Boutros, Jaewoong Sim, Phillip Tomson, Huseyin Sumbul, Gregory Chen, Phil Knag, Raghavan Kumar, et al. 2019. Why compete when you can work together: Fpga-asic integration for persistent rnns. In *Proceedings of the IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'19)*. IEEE, 199–207.
[52] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, Srivatsan Krishnan, and Debbie Marr. 2016. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. In *Proceedings of the 26th International Conference on Field Programmable Logic and Applications (FPL'16)*. IEEE, 1–4.

[53] Young H. Oh, Seonghak Kim, Yunho Jin, Sam Son, Jonghyun Bae, Jongsung Lee, Yeonhong Park, Dong Uk Kim, Tae Jun Ham, and Jae W. Lee. 2021. Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'21)*. IEEE, 584–597.

[54] Naebeom Park, Yulhwa Kim, Daehyun Ahn, Taesu Kim, and Jae-Joon Kim. 2020. Time-step interleaved weight reuse for LSTM neural network computing. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 13–18.

[55] Lu Peng, Wentao Shi, Jian Zhang, and Samuel Irving. 2019. Exploiting model-level parallelism in recurrent neural network accelerators. In *Proceedings of the IEEE 13th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC'19)*. IEEE, 241–248.

[56] Raghu Prabhakar, Yaqi Zhang, David Koeplinger, Matt Feldman, Tian Zhao, Stefan Hadjis, Ardavan Pedram, Christos Kozyrakis, and Kunle Olukotun. 2017. Plasticine: A reconfigurable architecture for parallel patterns. In *Proceedings of the ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA'17)*. IEEE, 389–402.

[57] Zhiqiang Que et al. 2020. Optimizing reconfigurable recurrent neural networks. In *Proceedings of the IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM'20)*. IEEE.

[58] Zhiqiang Que, Yanyang Liu, Ce Guo, Xinyu Niu, Yongxin Zhu, and Wayne Luk. 2019. Real-time anomaly detection for flight testing using AutoEncoder and LSTM. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'19)*. IEEE, 379–382.

[59] Zhiqiang Que, Hiroki Nakahara, Hongxiang Fan, Jiuxi Meng, Kuen Hung Tsoi, Xinyu Niu, Eriko Nurvitadhi, and Wayne Luk. 2020. A reconfigurable multithreaded accelerator for recurrent neural networks. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'20)*. IEEE, 20–28.

[60] Zhiqiang Que, Hiroki Nakahara, Eriko Nurvitadhi, Andrew Boutros, Hongxiang Fan, Chenglong Zeng, Jiuxi Meng, Kuen Hung Tsoi, Xinyu Niu, and Wayne Luk. 2022. Recurrent neural networks with column-wise matrix-vector multiplication on FPGAs. *IEEE Trans. VLSI Syst.* (2022).

[61] Zhiqiang Que, Thomas Nugent, Shuanglong Liu, Li Tian, Xinyu Niu, Yongxin Zhu, and Wayne Luk. 2019. Efficient weight reuse for large LSTMs. In *Proceedings of the IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP'19)*, Vol. 2160. IEEE, 17–24.

[62] Zhiqiang Que, Erwei Wang, Umar Marikar, Eric Moreno, Jennifer Ngadiuba, Hamza Javed, Bartłomiej Borzyszkowski, Thea Aarrestad, Vladimir Loncar, Sioni Summers, Maurizio Pierini, Peter Y. Cheung, and Wayne Luk. 2021. Accelerating recurrent neural networks for gravitational wave experiments. In *Proceedings of the 32th International Conference on Application-specific Systems, Architectures and Processors (ASAP'21)*. IEEE.

[63] Zhiqiang Que, Yongxin Zhu, Hongxiang Fan, Jiuxi Meng, Xinyu Niu, and Wayne Luk. 2020. Mapping large LSTMs to FPGAs with weight reuse. *J. Sign. Process. Syst.* 92, 9 (2020), 965–979.

[64] Stefano Ribes, Pedro Trancoso, Ioannis Sourdis, and Christos-Savvas Bouganis. 2020. Mapping multiple LSTM models on FPGAs. In *Proceedings of the International Conference on Field-Programmable Technology (ICFPT'20)*. IEEE, 1–9.

[65] Michalis Rizakis, Stylianos I. Venieris, Alexandros Kouris, and Christos-Savvas Bouganis. 2018. Approximate FPGA-based LSTMs under computation time constraints. In *Proceedings of the International Symposium on Applied Reconfigurable Computing*. Springer, 3–15.

[66] Vladimir Rybalkin, Alessandro Pappalardo, Muhammad Mohsin Ghaffar, Giulio Gambardella, Norbert Wehn, and Michaela Blott. 2018. FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL'18)*. IEEE.

[67] Vladimir Rybalkin, Chirag Sudarshan, Christian Weis, Jan Lappas, Norbert Wehn, and Li Cheng. 2020. Efficient hardware architectures for 1D-and MD-LSTM networks. *J. Sign. Process. Syst.* 92, 11 (2020), 1219–1245.

[68] Vladimir Rybalkin and Norbert Wehn. 2020. When massive GPU parallelism ain't enough: A novel hardware architecture of 2D-LSTM neural network. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[69] Vladimir Rybalkin, Norbert Wehn, Mohammad Reza Yousefi, and Didier Stricker. 2017. Hardware architecture of bidirectional long short-term memory neural network for optical character recognition. In *Proceedings of the Conference on Design, Automation & Test in Europe*. 1394–1399.

[70] Runbin Shi, Junjie Liu, K.-H. Hayden So, Shuo Wang, and Yun Liang. 2019. E-LSTM: Efficient inference of sparse LSTM on embedded heterogeneous system. In *Proceedings of the 56th ACM/IEEE Design Automation Conference (DAC'19)*. IEEE, 1–6.

[71] Gil Shomron, Tal Horowitz, and Uri Weiser. 2019. SMT-SA: Simultaneous multithreading in systolic arrays. *IEEE Comput. Arch. Lett.* 18, 2 (2019), 99–102.

[72] Franyell Silfa, Jose Maria Arnau, and Antonio Gonzalez. 2020. E-BATCH: Energy-efficient and high-throughput RNN batching. *ACM Transactions on Architecture and Code Optimization (TACO)* 19, 1 (2020), 1–23.

[73] Burton J. Smith. 1986. A pipelined, shared resource MIMD computer. In *Advanced Computer Architecture*. 39–41.
[74] Yuxi Sun, Akram Ben Ahmed, and Hideharu Amano. 2019. Acceleration of deep recurrent neural networks with an FPGA cluster. In *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*. 1–4.
[75] Zhanrui Sun, Yongxin Zhu, Yu Zheng, Hao Wu, Zihao Cao, Peng Xiong, Junjie Hou, Tian Huang, and Zhiqiang Que. 2018. FPGA acceleration of LSTM based on data for test flight. In *Proceedings of the IEEE International Conference on Smart Cloud (SmartCloud'18)*. IEEE, 1–6.
[76] Tian Tan, Eriko Nurvitadhi, David Shih, and Derek Chiou. 2018. Evaluating the highly-pipelined intel stratix 10 FPGA architecture using open-source benchmarks. In *Proceedings of the International Conference on Field-Programmable Technology (FPT'18)*. IEEE, 206–213.
[77] James E. Thornton. 1964. Parallel operation in the control data 6600. In *Proceedings of the Fall Joint Computer Conference, Part II: Very High Speed Computer Systems*. 33–40.
[78] Stylianos I. Venieris and Christos-Savvas Bouganis. 2018. f-CNNx: A toolflow for mapping multiple convolutional neural networks on FPGAs. In *Proceedings of the 28th International Conference on Field Programmable Logic and Applications (FPL'18)*. IEEE, 381–3817.
[79] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 3156–3164.
[80] Shuo Wang, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, Yanzhi Wang, and Yun Liang. 2018. C-LSTM: Enabling efficient LSTM using structured compression techniques on FPGAs. In *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 11–20.
[81] Zhisheng Wang, Jun Lin, and Zhongfeng Wang. 2017. Accelerating recurrent neural networks: A memory-efficient approach. *IEEE Trans. VLSI Syst.* 25, 10 (2017), 2763–2775.
[82] Zhao Wang, Guangyu Sun, Jingchen Zhu, Zhe Zhou, Yijiang Guo, and Zhihang Yuan. 2021. METRO: A software-hardware co-design of interconnections for spatial DNN accelerators. arXiv:2108.10570 [cs.AR]. Retrieved from https://arxiv.org/abs/2108.10570.
[83] Jiaquan Wu, Feiteng Li, Zhijian Chen, and Xiaoyan Xiang. 2019. A 3.89-GOPS/mW scalable recurrent neural network processor with improved efficiency on memory and computation. *IEEE Trans. VLSI Syst.* 27, 12 (2019), 2939–2943.
[84] Xilinx. 2017. Deep Learning with INT8 Optimization on Xilinx Devices. Retrieved from https://www.xilinx.com/support/documentation/whitepapers/wp486-deep-learning-int8.pdf.
[85] Krishna Praveen Yalamarthy et al. 2019. Low-complexity distributed-arithmetic-based pipelined architecture for an LSTM network. *IEEE Trans. VLSI Syst.* 28, 2 (2019), 329–338.
[86] Reza Yazdani, Olatunji Ruwase, Minjia Zhang, Yuxiong He, Jose-Maria Arnau, and Antonio González. 2019. LSTM-sharp: An adaptable, energy-efficient hardware accelerator for long short-term memory. arXiv:1911.01258. Retrieved from https://arxiv.org/abs/1911.01258.
[87] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. 2015. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4694–4702.
[88] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. arXiv:1409.2329. Retrieved from https://arxiv.org/abs/1409.2329.
[89] Tian Zhao, Yaqi Zhang, and Kunle Olukotun. 2019. Serving recurrent neural networks efficiently with a spatial accelerator. *Proc. Mach. Learn. Syst.* 1 (2019), 166–177.
[90] Yong Zheng, Haigang Yang, Yiping Jia, and Zhihong Huang. 2021. PermLSTM: A high energy-efficiency LSTM accelerator architecture. *Electronics* 10, 8 (2021), 882.

Received 2 September 2021; revised 18 February 2022; accepted 1 May 2022