A Reconfigurable Multithreaded Accelerator for Recurrent Neural Networks

Zhiqiang Que*, Hiroki Nakahara†, Hongxiang Fan*,
Jiuxi Meng*, Kuen Hung Tsoi§, Xinyu Niu§, Eriko Nurvitadhi‡, Wayne Luk*
† Tokyo Institute of Technology, Japan, nakahara.h.ad@m.titech.ac.jp
§ Corerain Technologies Ltd., China, {kuenhung.tsoi, xinyu.niu}@corerain.com
‡ Intel Corporation, eriko.nurvitadhi@intel.com
*Imperial College London, UK, {z.que, h.fan17, jiuxi.meng16, w.luk}@imperial.ac.uk

2020 International Conference on Field-Programmable Technology (ICFPT) | 978-1-6654-2302-1/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICFPT51103.2020.00012

***Abstract*—Recurrent Neural Network (RNN) is a key technology for sequential applications which require efficient and real-time implementations. Despite its popularity, efficient acceleration for RNN inference is challenging due to its recurrent nature and data dependencies. This paper proposes a multi-threaded neural processing unit (NPU) for RNN/LSTM inferences to increase processing abilities and quality of service of cloud-based NPUs by improving their hardware utilization. Besides, a custom coarse-grained multi-threaded LSTM (CGMT-LSTM) hardware architecture is introduced, which switches tasks among threads when LSTM computational kernels meet data hazard. These logical NPUs share nearly all resources of the physical NPU. When one logical NPU is stalled, another one can make progress. These optimizations improve the exploitation of parallelism to increase hardware utilization and enhance system throughput. Evaluation results show that a dual-threaded CGMT-LSTM NPU gains 27% more performance while only has 3.8% more area than a single-threaded one using a Stratix 10 FPGA. When compared with an implementation on the Tesla V100 GPU, our novel hardware architecture is 6.62 times faster and 15.88 times higher power efficiency, which demonstrates that our approach contributes to high performance energy-efficient FPGA-based multi-LSTM inference systems.**

I. Introduction

Recurrent Neural Network (RNN) is the main component of smart applications with sequential inputs, such as natural language processing [1], speech recognition [2, 3] and video analysis [4, 5]. Among the many RNN variants, Long Short-Term Memory (LSTM) is the most popular one. Since low-latency is key for a seamless user experience in such applications, efficient and real-time RNN/LSTM acceleration is required. FPGAs have been used to speed up the inference of LSTM [6, 7, 8, 9, 10], which offer benefits of low latency and low power consumption compared to CPUs or GPUs.

However, existing LSTM accelerators cannot support a cost-effective multi-recurrent neural network 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 center, CNNs comprise only 5% of the workload, while the Long Short-Term Memory (LSTM) make up 29% [11]. But most of the existing LSTM accelerators are only able to perform one task at a time. They can naively process the multi-LSTM tasks by executing different neural networks or layers in sequence, resulting in inefficiency when multiple task requests

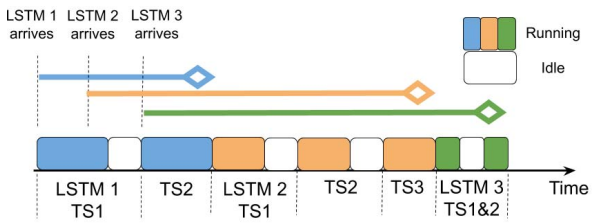


Fig. 1. Timeline of three LSTM inference tasks based on a first-come first-serve scheduling policy. The example of the LSTM 1 has two time-steps (TS), the LSTM 2 has three time-steps and the LSTM 3 has two time-steps in this figure.

come at the same time, as shown in Fig. 1. It may make the latter coming task wait for a long time to be processed with an available hardware core since the former LSTM layer may have a large number of time-steps which needs lots of iterations, e.g., 1500 time-steps in an LSTM layer in DeepSpeech [12]. Besides, some specific applications involve multiple LSTM models. A spacecraft anomalies detection system [13] even involves over 700 LSTM models, each modeling a single telemetry channel and predicting values for that channel, which shows the need to support multi-LSTM execution. Moreover, a conventional LSTM accelerator is often designed by arranging all computing resources to form a single core with a large scale, leveraging data level parallelism. For example, Brainwave [9] is a single-threaded neural processing unit (NPU) which has 96,000 processing elements (PEs). However, when the size of the targeted LSTM model is small, these hardware resources will not be fully utilized, e.g., the Brainwave hardware utilization is lower than 1% [9] and the utilization of NPU in [10] is lower than 15% when targeting a small LSTM model (h_t=256). It is challenging to design an accelerator to support a cost-effective multi-LSTM execution.

This paper proposes a multi-threaded neural processing unit (NPU) for accelerating LSTM inferences to increase the processing abilities of cloud-based NPUs with high quality of service and improved hardware utilization for better performance. A primary goal of our design is to efficiently add more per-NPU core scalability potential. The most area / cost efficient way to add "logical" cores is multithreading. Essentially, multithreading 'recovers' unused performance (where

978-1-6654-2302-1/20/$31.00 ©2020 IEEE
DOI 10.1109/ICFPT51103.2020.00012

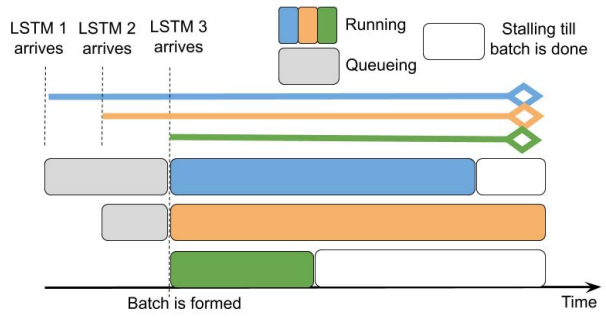


Fig. 2. Timeline of three LSTM inference tasks using batching

execution units are idle due to events on one thread) by switching to another thread. Multithreading also ensures there is no downside to peak performance in single-threaded mode. Running multiple neural networks together has a potential to alleviate the issues of idle as layers from different neural networks can be freely scheduled without any dependency issue. The execution of multiple tasks can also be achieved by a batch technique which feeds multiple inputs into a neural network to generate multiple outputs in one inference round. However, it harms latency since different inputs may not come at the same time [14], meaning that a newly arrived request have to wait until the batch is formed, which imposes significant latency penalty, as shown in Fig. 2.

In addition, inspired by coarse-grained multithreading used in modern CPUs, e.g., IBM RS64-IV [15] and Intel Montecito [16], which makes a core switch to a different hardware context when a thread is stalled due to some events, we propose a custom coarse-grained multi-threaded LSTM (CGMT-LSTM) hardware architecture which switches tasks among threads when LSTM computational kernels meet data hazard. When one logical NPU core is stalled, the other can make progress. The Coarse-grained multithreading (CGMT) is a mature technique in modern CPU designs. However, few studies concern combining the CGMT and NPUs, especially for RNNs/LSTMs. Unlike CNNs which do not have memory cells and can run layers from different neural networks iteratively, RNNs/LSTMs have memory cells, which makes it hard to run different timesteps from different RNN layers or models since they have different cell memory using a single-threaded accelerator. It needs to finish the former model or former RNN layer until the next model or layer can run. The existence of inter-timestep dependency within a RNN model prevents the following timesteps from even starting their execution until the current timestep's completion, which leads to a hardware underutilization. To address this challenge, we propose the CGMT-LSTM which can rapidly switch to an alternate thread of computation during a data-hazard event, e.g. the inter-timestep dependencies of RNNs, to achieve high hardware utilization and improve the system performance.

To the best of knowledge, this is the first work to propose an coarse-grained multi-threaded LSTM accelerator architecture to enable an effective multi-LSTM execution.

We make the following contributions in this paper:

- A novel multi-threaded neural processing unit to enable effective multi-neural network execution for LSTMs.
- A custom coarse-grained multi-theaded LSTM hardware architecture which significantly improves hardware utilization and system performance.
- A custom tiling method of LSTMs, which minimizes the intermediate results buffers when combining the CGMT, thereby increasing the accelerator area efficiency.
- A comprehensive evaluation of the proposed method and hardware architecture.

II. Background and Preliminaries

LSTMs are artificial neural networks which have feedback connections and internal memory cells to record past information about long-term dependencies over an arbitrary time. They achieve high accuracy in many sequence processing problems such as text analysis, speech recognition and video classification.

LSTM was initially proposed in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [17]. This study follows the standard LSTM cell [7, 9, 10, 5], as shown in Fig 3. The hidden state h_t is produced by the following equations:

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

Here, σ, $tanh$ and $\odot$ stand for the sigmoid function, the hyperbolic tangent function and element-wise multiplication respectively. i, f, g and o represent the input, forget, input modulation and output gate respectively. The input modulation gate is often considered as a sub-part of the input gate. The input vector and hidden vector are combined so that W represents the weight matrix for both input and hidden units. Bias is represented as b. The output c_t is the internal memory cell state and h_t is the output of the cell, also called the hidden state, which is passed to the next time-step or next layer. The gates control the information flow inside the LSTM unit. The input gate decides what new information is to be written into the memory cell; the forget gate decides what old information is no longer needed and can be removed; the input modulation gate is used to modulate the information that the input gate will write into the memory cell by adding non-linearity to the information; the output gate decides what the next hidden state should be. Our work focuses on the optimization of the standard LSTM, but the proposed techniques can be applied to other RNN and LSTM variants.

III. Design and Optimization Methodology

It is challenging to accelerate RNN/LSTM inference efficiently due to its recurrent nature and data dependencies. In this section, we first introduce the coarse-grained multi-threading for accelerating RNN/LSTM. We then propose a

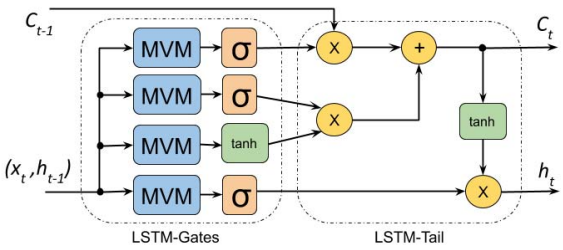


Fig. 3. Structure of an LSTM Cell

TABLE I
SYSTEM PARAMETERS

W	Weights matrix
Hw	Number of columns of weight matrix
Lw	Number of Rows of weight matrix
x_t	The input vector x at timestep t
h_t	The hidden vector h at timestep t
Lx	Number of elements in input vector x
Lh	Number of elements in hidden vector h
NPE	Number of processing elements
EP	Element-based Parallelism
VP	Vector-based Parallelism
TS	Timestep

blocking strategy of LSTM weights matrix to enable sub-layer granularity scheduling to create fine-grained tasks for CGMT-LSTM architecture. We define the system parameters in Table I which are used for later calculations.

A. Multithreading for Recurrent Neural Processing Unit

There are huge demands for architectural support of multi-DNN running to maximize the hardware utilization and reduce the cost of running large-scale production systems. However, most of the existing LSTM accelerators are only able to run one task at a time. This paper proposes a coarse-grained multi-threaded (CGMT) LSTM NPU which switch on the event of computational unit data hazard in LSTM operations. The general idea is when a thread is stalled due to some event, e.g cache misses, the cores can switch to a different hardware context. In our CGMT-LSTM NPU, the event is the data hazard due to data dependency between the sequence time-step calculation in LSTMs. The idea is to have multiple thread contexts in a single recurrent neural processor unit so that when the first thread stalls then the second one can continue to run, as shown in Fig. 4. Thus, it can utilize processing resources more efficiently to improve system performance by exploiting thread-level parallelism and improving NPU utilization.

A former LSTM model or layer may have thousands of time-steps, which occupies the processor for a long time, resulting in long waiting time for the latter requests before they can be executed. However, some services are latency critical because their response time directly relates to user experience,

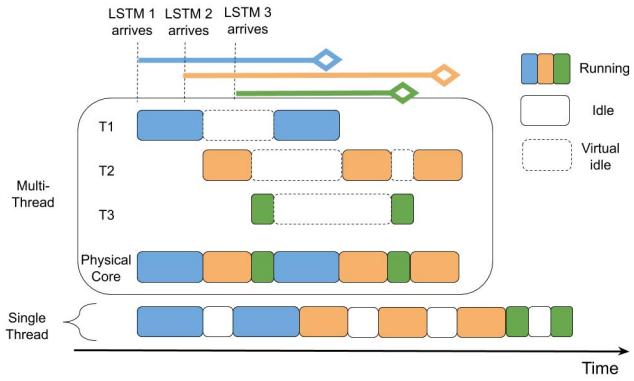


Fig. 4. Timeline of three LSTM inference tasks based the proposed CGMT

e.g. intelligent personal assistants are one of the examples where real-time deep learning is used to process user speech and give smart responses. The conventional single thread NPU can write back the old thread's context to memory and load the new thread's context using preemption mechanisms [18] to run another task. However, it will bring a large context switch penalty. In our new multi-threaded NPU for LSTMs, the new task can be executed from another thread as soon as it comes, as shown in Fig. 4. Please note that the particular thread may still stall because of data hazard, but the physical core is not stalled since multiple threads share the same computational physical core, which results in "Virtual idle" as shown in this figure. We believe that further optimizations, e.g. Simultaneous Multi-Threading (SMT) can be applied to our NPU implementation to achieve even higher performance. We leave that for future work since it has a limited impact on the conclusions we draw from our study in this paper.

B. Weights Matrix blocking Strategy

According to the LSTM equations (1), an LSTM layer of a single time-step involves four matrix-vector multiplication operations, which are independent. Since the four matrices of i, f, o, u gates of LSTMs share the same size, we combine these matrices into one large matrix [19, 20] as shown in Fig. 5. Thus, in one time-step of the LSTM operation, we only need to focus on optimizations of one large matrix multiplied by one vector for the whole LSTM cell instead of four small matrices multiplying one vector. This is a generic optimization that can be applied to any Matrix-Vector Multiplications (MVMs) that share the same input vector. Since each gate matrix has the size of $Lh \times (Lx + Lh)$, the large combined matrix has the size of $(4 \times Lh) \times (Lx + Lh)$.

Generally, the weight matrices in neural networks are large and they can be accelerated by operating in parallel. However, the resources on FPGAs are limited, which means that we cannot perform the whole MVM computation at once, especially for some large LSTMs. In order to utilize the computational engine efficiently, we partition the combined weights matrix of an LSTM layer into multiple sub-layers in advance depending on the detailed configuration of accelerator and LSTM sizes.

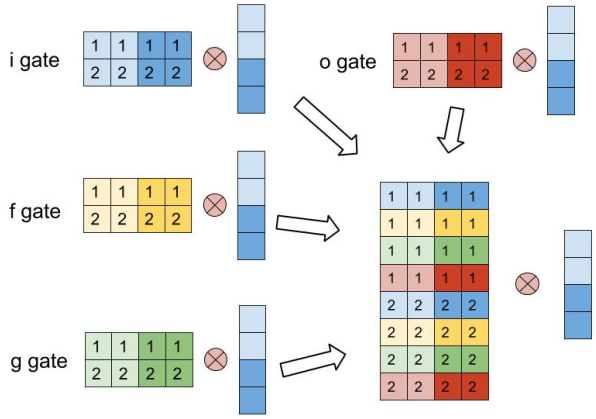
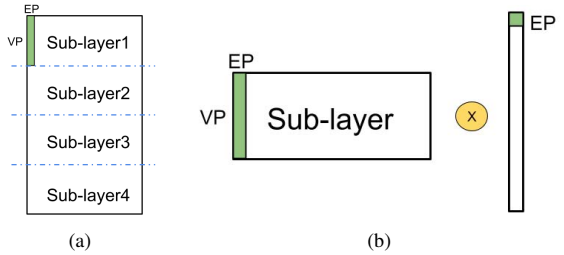


Fig. 5. The gates weights matrices and the combined weights matrix, showing interleaving of four gates weights. The lengths of the input vector and hidden vector are 2 in this example.



(a) (b)

Fig. 6. The weights matrix blocking. (a) The combined weights matrix, showing the sub-layers. The number of sub-layers are 4 in this example. (b) One sub-layer using the Element-based Parallelism (EP) and Vector-based Parallelism (VP) as a tile shaded in green

Specifically, an LSTM layer of one time-step is firstly divided into a number of equal-sized sub-layers, as shown in Fig. 6a. To illustrate the idea, the number of the sub-layers in this figure is shown as four, however, the real system can have more sub-layers. Then, Element-based Parallelism (EP) and Vector-based Parallelism (VP) are introduced to exploit the available parallelism [20]. The number of the sub-layers equals $\frac{(4\times Lh)}{VP}$. The sub-layer is then partitioned into several small tiles which have a size of (EP, VP), as shown in Fig. 6b. In each cycle, our CGMT-LSTM core can process a tile of the sub-layer and a sub-vector of $[x_t, h_{t-1}]$ with a size of EP. To increase system parallelism, VP is chosen to be as large as possible. However, the largest number of VP is Hw, which equals $4 \times Lh$, since there are only four gates in LSTM. Thus, the smallest number of sub-layers is one. In this work, sub-layer granularity scheduling is adopted to create fine-grain tasks for multi-LSTM execution.

By interleaving the rows of the weights matrices, the related elements in the result vector from four gates are adjacent and can be reduced easily via the element-wise operations in the LSTM-tail unit. It means there are no data dependencies between these sub-layers. The technique of interleaving also removes the need for buffering large intermediate outputs from four LSTM gates since the result sub-vectors of various sub-layers are not related and will be reduced in the tail unit soon. Each sub-vector of the MVM result can be processed individually in the LSTM-tail units. There is no need to wait for other sub-layer results to achieve the LSTM memory cell status and hidden units of the current sub-layer. The proposed CGMT-LSTM core will always finish processing all the sub-layers in one time-step before it switches to another thread since the data hazard happens when time-step changes. Compared with a fine-grained multithreading scheme which switches the context between threads in each cycle, we avoid to buffer these large intermediate MVM operations values as well as element-wise operations values since these values will finally form the sub-vector of LSTM memory cells and hidden units. We only need to add thread dedicated buffers for these LSTM memory cells and hidden units since different threads process different LSTM requests.

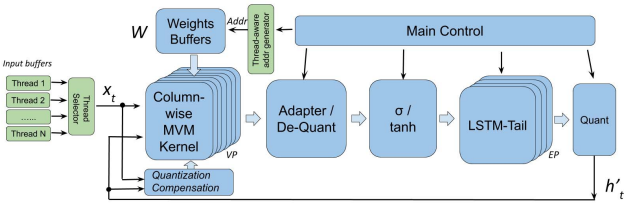


Fig. 7. The overview of the system

IV. Hardware Architecture

Based on the optimization techniques introduced above, we implement the proposed CGMT-LSTM scheme on top of a state-of-the-art RNN accelerator [20] for low latency cloud-based applications. In the Section IV-A., we outline the main components of the architecture and detail the necessary hardware modifications required in order to support our CGMT-LSTM scheme. A few FPGA-specific optimizations are also introduced.

A. System details

A high-level block diagram of this accelerator is shown in Fig. 7. It is composed of VP kernel units, an Adapter / De-Quant unit, an activation function unit, EP tail units, and a Quant unit as well as data buffers. The proposed multi-threaded NPU has many logical neural processing units which share nearly all resources of the physical NPU, e.g. weights buffer, kernel units, activation function units, element-wise units.

Each kernel has EP Processing Elements (PEs), resulting in $VP \times EP$ effective PEs. The VP and EP values are determined via the design space exploration described in detail in [20]. In this design, 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}. In our proposed CGMT-LSTM NPU, all the logical cores share all the kernel units. When one logical core (thread) is stalled, the other can still perform calculation using the same kernel

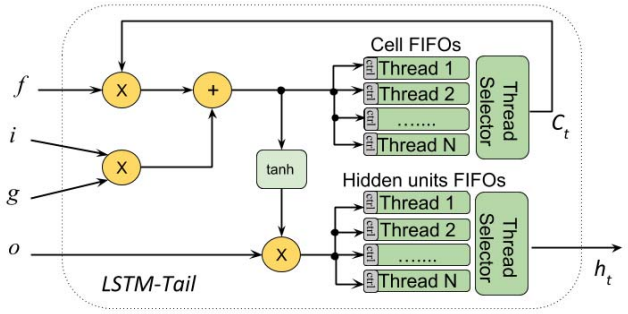


Fig. 8. The LSTM-tail unit

units. Thus, the proposed CGMT-LSTM NPU is able to utilize the kernel resources efficiently to increase system performance and improve NPU hardware utilization. Since each thread requires its own input vector x_t and h_{t-1}, the system has to keep multiple buffers to store these contexts. Besides, thread selection logic is necessary to choose the required buffer to retrieve the data and perform the calculations, as shown in Fig. 7.

The adapter converts the parallelism between kernels and tails. Practically, the design does not require a large parallelization factor in the tail units as in the kernel units since the column-wise MVM produces one output vector after multiple accumulation cycles. Then, de-quantization (De-Quant) is applied to convert quantized values into fixed-point values to reduce hardware resources. Unlike the kernel unit which mainly contains dot product operations, the tail unit is composed of a sequence of element-wise operations, which can introduce frequent de-quantization and quantization operations in the linear-quantization scheme. Since de-quantization and quantization require a 32-bit multiplier unit and an adder, the total hardware cost of linear-quantization for the tail unit is higher than that of 16-bit fixed-point quantization. Therefore, we do not perform linear-quantization in the tail units. Instead, the quantized values will be de-quantized into fixed-point values.

The activation function unit performs the sigmoid (σ) and hyperbolic tangent ($tanh$) functions. Both are implemented using software programmable lookup tables of size 2048 [3, 10]. The implementation using lookup tables has many benefits: (1) our neural processor is able to run a trained model using custom activation functions (e.g. the hard sigmoid in Keras [21]) from users without retraining. Since we do not change the calculation equations in the model and do not perform retraining, we do not touch users' sensitive data which are critical for many users; (2) a lookup table has a fixed latency of 1 cycle, while other implementation, e.g. a piece-wise linear approximation, will involve multipliers which may have a much larger latency.

The LSTM-tail units as shown in Fig. 8 mainly perform the element-wise operations. The LSTM memory cell FIFOs and hidden units FIFOs are used to store the status of the running LSTMs since one thread may be switched because of data hazard before it can fully calculate the current LSTM layer with multiple time-steps. Since these threads run different LSTM layers or models, the system has to maintain multiple hardware contexts, as shown in Fig. 8. The output hidden vector (h_t) needs the quantization (Quant) before it can be used in the MVM kernels, so a Quant unit is utilized after the final output of LSTM-tail units as shown in Fig. 7.

TABLE II
BENCHMARKS USED IN THIS WORK

Name	Length (h_t)	Length (x_t)	Domain
IMDB [28]	128	128	Sentiment Classification
LRCN [5]	256	2048	Activity Recognition
Show & Tell [29]	512	512	Image Caption Generation
DeepBench [12]	256 / 512 / 1024	256 / 512 / 1024	Speech Recognition

B. FPGA-Specific optimizations

Since the proposed CGMT-LSTM NPU can process a tile which has the size (EP, VP) in each cycle, all the kernel units can share the same input of a partial sub-vector of (x_t, h_{t-1}), which has EP elements. These EP elements are broadcasted to all these kernels. Besides, a single address needs to be broadcasted to all the weight buffers. To mitigate the large fan-out issue, the tree-shaped [10] interconnect is adopted to reduce 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 Stratix 10 HyperFlex registers to achieve higher operating frequency [22, 23].

The DSP blocks in modern FPGAs, which have high configurability, are often underutilized when implementing 8-bit DNN systems. Both [24] and [25] show a method to extract two 8-bit multipliers from one large precision multiplier in FPGA DSP blocks. In this design, the approach in [25] is adopted to pack four 8-bit into one DSP block on Intel FPGAs to reduce the hardware resources. Besides, this would not be a restriction (and will come at lower cost) if we use a novel DSP similar to what was proposed in [26] and will be adopted in the next generation Agilex devices [27].

V. Evaluation and Analysis

This section presents hardware implementation results using an Intel Stratix 10 FPGA that demonstrate the scalability of the proposed optimizations for RNNs.

A. Experimental Setup

For our study, we choose our benchmark workloads from a few typical LSTM applications which are listed in Table II for a fair and direct comparison. Multi-tasked LSTM workloads are constructed randomly from these LSTMs. An Intel Stratix 10 2800 (S10) is evaluated and compared with other work. It runs the inferences of persistent LSTM models which store the weights in on-chip memory [9, 10, 20]. It is designed using Verilog RTL. Quartus Prime Pro 19.4 is used to target S10.

TABLE III
RESOURCE UTILIZATION

	Thread	ALMs	M20K	DSP	Freq.
Stratix 10 (2800)	Single	487,232 (52.2%)	10,061 (85.8%)	4,368 (76%)	260 MHz
Stratix 10 (2800)	Dual	522,852 (56.0%)	10,129 (86.4%)	4,368 (76%)	260 MHz

TABLE IV
PERFORMANCE COMPARISON OF THE FPGA DESIGN VERSUS CPU AND GPU

	CPU	GPU	This work
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 (h_t)	1024		
TDP Power(W)	15	300	125
Throughput (GOPS)	8	1180	7810
Power Efficiency (GOPS/W)	0.53	3.93	62.48

The EP is 16 and VP is 1024 according to [20]. The number of PEs, NPE, is 16384.

B. Resource Utilization

Table III shows the resource utilization of our designs with two configurations on FPGAs. Both the designs utilize the parameter of (EP, VP) as (16, 1024), which include 16,384 8-bit multipliers in the MVM kernels implemented using DSP blocks with extra ALMs. The dual-threaded design consumes 3.8% more of total logic (ALMs) resources and 0.6% more block ram (M20K) than the single-threaded one. Since dual threads share the same physical core it consumes the same DSP resources with the single-threaded one.

C. Performance and Efficiency Comparison

To compare the performance of the proposed design on FPGA with other platforms, the DeepBench published results [30] on an Intel Xeon Syklake CPU and NVIDIA Tesla V100 GPU are used. The AVX2 vector instructions are enabled for the CPU while the CuDNN libraries are enabled for the GPU. Both CPU and GPU implementations run with a batch size of 1 which provides the lowest cloud service latency since requests need to be processed as soon as they arrive. For a fair comparison with the throughput of our dual-threaded CGMT-LSTM processing unit, the throughput of the CPU and GPU have been doubled in Table IV. Compared with the RNN running on GPU, our FPGA design of CGMT-LSTM is 6.62 times faster and 15.88 times higher power efficiency respectively as shown in Table IV.

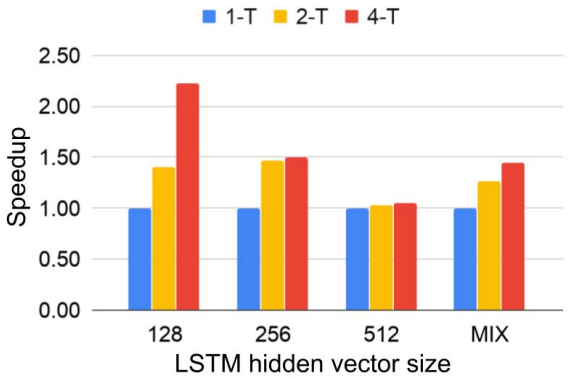


Fig. 9. Performance speedup using various threads.

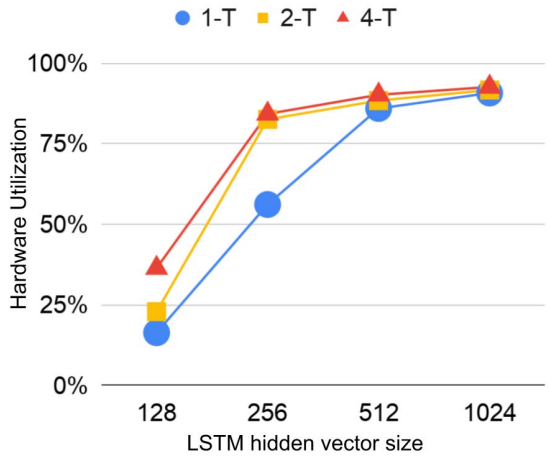


Fig. 10. Hardware utilization of different LSTM tasks using various threads.

To illustrate the benefits of our proposed approach, we compare the proposed multi-threaded LSTM processing unit with a single thread core in Fig. 9 and Fig. 10. Hardware utilization is the percentage of the peak Tera Operations Per Second (TOPS) reached for each layer. With the proposed CGMT-LSTM approach, the LSTM designs with 2 threads (2-T) and 4 threads (4-T) achieve 1.27 and 1.45 times higher performance respectively than the baseline [20] using a single thread core when targeting mixed requests from LSTM workloads with $h = 128, 256$ and 512. With 4-T, the system achieves 2.23 times higher performance than the baseline [20] when targeting the small LSTMs. The performance gain is slight when only targeting large LSTMs since the baseline LSTM processing unit [20] already achieves high hardware utilization for these LSTMs. However, the baseline still suffers from low utilization when targeting small sized LSTMs which are commonly used in many applications [31, 5]. LSTM models with small sizes that have large number of time-steps are the most tangible examples that require dealing with lots

of dependencies, as well as the parallel task of MVMs [32]. Our proposed approach and hardware architecture can alleviate this problem by leveraging the coarse-grained multithreading.

Our experiments also show that the utilization is low when targeting a small LSTM ($h = 128$). It is because the largest effective VP in this case is $4 \times Lh$ which is 512 so that the effective PEs are less than NPE, which leads to severe underutilization. Different choices of EP and VP mapping impact the performance and utilization of systems running LSTM models with different sizes. It is a trade-off between the extra performance which can be gained and the design complexity as well as extra hardware resources for supporting various values of (EP, VP) dynamically. It is our future work to improve this architecture using an overlay to support various values of (EP, VP) since different model sizes may prefer different best EP and VP parameters.

Some existing FPGA-based RNN/LSTM accelerator designs are compared with ours in Table V. For a fair comparison, we only show the previous work with a detailed implementation of the LSTM system using the Stratix 10 GX2800. We show the FPGA chips, model storage, precision, DSP used, runtime frequency, throughput and power efficiency and hardware (HW) utilization. The thermal design power (TDP) is used for a fair comparison since it is reported in all mentioned papers. Overall, our design provides over 4.92 times higher performance and 5.77 times higher hardware utilization than the state-of-the-art design [10] when targeting an LSTM model (h_t=256), as shown in Table V. When compared with the single-threaded NPU [20], it achieves 1.47 higher performance and utilization. Since multithreading also ensures there is no downside to peak performance in single-threaded mode, the CGMT-LSTM can achieve the same perk performance of 8015 GOPS with the one in [20], which is the highest with respect to state-of-the-art FPGA-based RNN design using commonly used INT8 precision. The only prior work that provides a higher perk throughput is [9] using 8-bit block floating-point. However when targeting the small LSTM model, the throughput of the proposed CGMT-LSTM is 19 times higher than [9]. Furthermore, we achieve the highest hardware utilization among all these designs across various LSTM models. This paper focuses on minimizing latency and maximizing throughput by increasing the hardware utilization. The proposed architecture improves the efficiency of hardware usage from another point of view, hence it accelerates the effective performance.

TABLE V
COMPARISON WITH PREVIOUS IMPLEMENTATIONS OF LSTM USING STRATIX 10 GX 2800

	ISCA18-BW [9]	FCCM19-NPU [10]	FCCM20-NPU [20]	This work[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
LSTM Size (h_t)	256			
Performance (GOPS)	370	1431	4790	7041
Power[b] Effi. (GOPS/W)	2.96	11.45	38.32	56.33
LSTM HW Utilization	0.8%	14.3%	56.1%	82.5%

[a] Dual-threaded mode

[b] TDP Power is used

VI. RELATED WORK

There has been much previous work on FPGA-based implementations of persistent LSTM whose weights are stored in on-chip memory [9, 10, 33, 34, 35]. Besides, FINN-L [34] employs 1-8 bits as the quantized implementation which surpasses a single-precision floating-point accuracy for a given dataset. There are also some previous studies of LSTM implementations storing weights in off-chip memory on FPGA devices [7, 36, 37, 38, 39], which had been identified as the performance bottleneck. In addition, some novel LSTM weights reuse methods [39, 38, 40] are proposed to reduce the off-chip memory access to decrease the energy cost and improve the system throughput. Some previous work is trying to optimize LSTM inference using block-circulant-matrix-based approach [41, 42]. In [14, 34, 43, 44], the batching technique is proposed to increase the throughput of LSTM inferences. [44] proposes E-BATCH for RNNs which increases throughput while also improving energy efficiency on an ASIC-based accelerator. Furthermore, a framework which deploys an approximate computing scheme for LSTMs using small tiles is presented in [45]. Sun et al. [46] propose a multi-FPGA based approach for accelerating deep RNNs which achieves single-layer speed for arbitrarily deep RNN architectures with an FPGA cluster. [47] introduces a multiple FPGA architecture for accelerating neural machine translation. [48] explores RNN partitioning strategies to achieve the scalable multi-FPGA acceleration for large RNNs with analysis the performance impact of collective communications and software pipelining.

Some of the previous studies [3, 49, 50, 51, 52] are focusing on weight pruning and model compression to reduce the size of weights to achieve good performance and efficiencies. In [3], the authors propose a pruning technique which compresses a large LSTM to fit the on-chip memory of an FPGA and improves inference efficiency. While in [49], DeltaRNN is proposed. It is based on the Delta Network algorithm which skips dispensable computations during inference of networks. The authors in [50] propose Bank-Balanced Sparsity (BBS) which is able to maintain model accuracy and enable an efficient FPGA accelerator implementation. In [53], BLINK is proposed which utilize bit-sparse data representation for the LSTM inference. It improves the energy efficiency of the LSTM inference by turning the multiplication into the bit shift operation while remaining the inference accuracy. These studies are orthogonal to our proposed approach and hardware architecture. These techniques can be complementary to our

approach to achieve even higher throughput of RNN inferences on FPGAs.

The Brainwave system [9] is a single-threaded SIMD architecture for persistent RNNs. It achieves more than an order of magnitude improvement in latency and throughput over GPUs on large memory intensive RNNs. Its idea is pinning the model weights into on-chip memory in order to achieve a necessary high memory read bandwidth to achieve high performance for RNNs. The authors in [30] propose the cross-kernel optimization within RNN cells targeting Plasticine [54] which is based on a coarse-grained reconfigurable architecture (CGRA). The authors in [10] introduce a Brainwave-like neural processing unit (NPU) for RNNs. They also propose TensorRAM for large persistent data intensive RNN sequence models. [20] proposes a novel latency-hiding hardware architecture based on fine-grained column-wise matrix-vector multiplication to eliminate data dependency, improving the throughput of systems of RNN/LSTM models. However, all these RNN/LSTM NPUs are single-threaded.

There are huge demands for architectural support of multi-DNN running to maximize the hardware utilization and reduce the cost of running large-scale production systems. For example, the concurrent DNN execution is supported in the TensorRT from NVIDIA for users to run multiple DNNs on the same GPUs simultaneously. SMT-SA [55] introduces a simultaneous multi-threading technique in Systolic Arrays to address the underutilization problem caused by zero-valued input. However, it cannot handle the underutilization problem caused by RNN data dependencies. In [56], A multi-threaded CGRA is proposed to accelerate CNN only. [57] proposes a synthesis technique to auto-generate in-order multi-threaded processing pipelined datapath from a high-level unpipelined datapath specification. In [18], the authors focus on a preemptive scheduling algorithm for a DNN accelerator to support multi-DNN running. But it does not explore the data dependencies between LSTM time-steps. The sequence length (time-step) of an LSTM model is usually larger or much larger than the number of layers. [58] proposes AI-MultiTasking, which balances compute- and memory-intensive tasks from different networks and executes them in parallel by dividing each layer into multiple identical sub-layers. It deals with RNNs just as FCs in its scheduling scheme. However, RNNs/LSTMs are more complex than FCs. In [31], the authors propose multithreading for LSTM accelerators based on a multi-core approach. However, they utilize the multi-core with each thread on each core. This affects the system performance in a single thread mode since they can only use half of the computational resources to run a single task. Thus, it is not an area / cost efficient way to add threads. In our design, a multi-threaded single-core accelerator is designed, which can run a task in a single thread mode to achieve higher performance and lower latency when targeting some high priority workloads while still handling multiple requests using multi-threaded mode if needed. A primary goal of our design is to efficiently add more per-NPU core scalability potential.

VII. Conclusions and Future Work

This paper proposes a coarse-grained multi-threaded neural processing unit (NPU) for LSTM inference to increase the processing abilities of cloud-based NPUs, increasing their throughput while improving their hardware utilization. We have implemented the proposed CGMT-LSTM architecture on Stratix 10 FPGAs with superior performance and efficiency, which shows the effectiveness of our approach. Further research includes combining our method with simultaneous multithreading (SMT) and the automation of the proposed approach to enable rapid development of efficient RNN/LSTM designs.

Acknowledgement

The support of the United Kingdom EPSRC (grant numbers EP/L016796/1, EP/N031768/1, EP/P010040/1, and EP/S030069/1), Corerain and Intel is gratefully acknowledged.

References

[1] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[2] D. Amodei *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, 2016.

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

[4] J. Yue-Hei Ng *et al.*, "Beyond short snippets: Deep networks for video classification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[5] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[6] E. Nurvitadhi *et al.*, "Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC," in *26th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2016.

[7] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*. IEEE, 2017, pp. 629–634.

[8] Z. Sun and et al., "FPGA acceleration of LSTM based on data for test flight," in *IEEE International Conference on Smart Cloud (SmartCloud)*, 2018, pp. 1–6.

[9] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*. IEEE Press, 2018, pp. 1–14.

[10] E. Nurvitadhi *et al.*, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs," in *27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019.

[11] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017.

[12] A. Hannun *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[13] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom, "Detecting spacecraft anomalies using lstm and nonparametric dynamic thresholding," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 387–395.

[14] P. Gao, L. Yu, Y. Wu, and J. Li, "Low latency RNN inference with cellular batching," in *Proceedings of the Thirteenth EuroSys Conference*, 2018, pp. 1–15.

[15] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel, "A multithreaded PowerPC processor for commercial servers," *IBM Journal of Research and Development*, vol. 44, no. 6, pp. 885–898, 2000.

[16] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-thread itanium processor," *IEEE micro*, vol. 25, no. 2, pp. 10–20, 2005.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] Y. Choi and M. Rhu, "PREMA: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020.

[19] M. S. Abdelfattah, D. Han, A. Bitar, R. DiCecco, S. O'Connell, N. Shanker, J. Chu, I. Prins, J. Fender, A. C. Ling *et al.*, "DLA: Compiler and FPGA Overlay for Neural Network Inference Acceleration," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 411–4117.

[20] Z. Que, H. Nakahara, E. Nurvitadhi, H. Fan, C. Zeng, J. Meng, X. Niu, and W. Luk, "Optimizing Reconfigurable Recurrent Neural Networks," in *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020, pp. 10–18.

[21] F. Chollet *et al.*, "Keras: Deep learning library for theano and tensorflow," *URL: https://keras. io/k*, vol. 7, no. 8, p. T1, 2015.

[22] T. Tan, E. Nurvitadhi, D. Shih, and D. Chiou, "Evaluating The Highly-Pipelined Intel Stratix 10 FPGA Architecture Using Open-Source Benchmarks," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 206–213.

[23] Intel, "Understanding how hyperflex architecture enables high performance systems," in *White Paper 01231*. Intel.

[24] "Deep Learning with INT8 Optimization on Xilinx Devices," 2017. [Online]. Available: https://www.xilinx.com/support/documentation/white papers/ wp486-deep-learning-int8.pdf

[25] M. Langhammer, B. Pasca, G. Baeckler, and S. Gribok, "Extracting INT8 Multipliers from INT18 Multipliers," in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2019.

[26] A. Boutros, S. Yazdanshenas, and V. Betz, "Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018, pp. 35–357.

[27] Intel, "Intel Agilex Variable Precision DSP Blocks User Guide," 2020.

[28] A. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[29] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3156–3164.

[30] T. Zhao, Y. Zhang, and K. Olukotun, "Serving Recurrent Neural Networks Efficiently with a Spatial Accelerator," *arXiv preprint arXiv:1909.13654*, 2019.

[31] L. Peng *et al.*, "Exploiting Model-Level Parallelism in Recurrent Neural Network Accelerators," in *International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2019.

[32] R. Yazdani, O. Ruwase, M. Zhang, Y. He, J.-M. Arnau, and A. González, "LSTM-Sharp: An Adaptable, Energy-Efficient Hardware Accelerator for Long Short-Term Memory," *arXiv preprint arXiv:1911.01258*, 2019.

[33] Z. Que *et al.*, "Real-time Anomaly Detection for Flight Testing using AutoEncoder and LSTM," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2019.

[34] V. Rybalkin, A. Pappalardo, M. M. Ghaffar, G. Gambardella, N. Wehn, and M. Blott, "FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs," in *28th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018.

[35] V. Rybalkin and N. Wehn, "When Massive GPU Parallelism Ain't Enough: A Novel Hardware Architecture of 2D-LSTM Neural Network," in *The 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2020, pp. 111–121.

[36] A. X. M. Chang, B. Martini, and E. Culurciello, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

[37] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE, 2017, pp. 152–159.

[38] N. Park, Y. Kim, D. Ahn, T. Kim, and J.-J. Kim, "Time-step interleaved weight reuse for LSTM neural network computing," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 13–18.

[39] Z. Que, T. Nugent, S. Liu, L. Tian, X. Niu, Y. Zhu, and W. Luk, "Efficient Weight Reuse for Large LSTMs," in *2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, vol. 2160. IEEE, 2019, pp. 17–24.

[40] Z. Que *et al.*, "Mapping Large LSTMs to FPGAs with Weight Reuse," *Journal of Signal Processing Systems*, 2020.

[41] S. Wang, Z. Li, C. Ding, B. Yuan, Q. Qiu, Y. Wang, and Y. Liang, "C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018, pp. 11–20.

[42] Z. Li *et al.*, "E-RNN: Design optimization for efficient recurrent neural networks in FPGAs," in *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019.

[43] A. Ardakani *et al.*, "Learning to skip ineffectual recurrent computations in lstms," *arXiv preprint arXiv:1811.10396*, 2018.

[44] F. Silfa, J. M. Arnau, and A. Gonzalez, "E-BATCH: Energy-Efficient and High-Throughput RNN Batching," *arXiv preprint arXiv:2009.10656*, 2020.

[45] M. Rizakis, S. I. Venieris, A. Kouris, and C.-S. Bouganis, "Approximate FPGA-based LSTMs under computation time constraints," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2018.

[46] Y. Sun *et al.*, "Acceleration of deep recurrent neural networks with an fpga cluster," in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, 2019.

[47] E. Nurvitadhi, A. Boutros, P. Budhkar, A. Jafari, D. Kwon, D. Sheffield, A. Prabhakaran, K. Gururaj, P. Appana, and M. Naik, "Scalable Low-Latency Persistent Neural Machine Translation on CPU Server with Multiple FPGAs," in *2019 International Conference on Field-Programmable Technology (ICFPT)*. IEEE, 2019, pp. 307–310.

[48] D. Kwon, S. Hur, H. Jang, E. Nurvitadhi, and J. Kim, "Scalable Multi-FPGA Acceleration for Large RNNs with Full Parallelism Levels," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[49] C. Gao *et al.*, "DeltaRNN: A power-efficient recurrent neural network accelerator," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2018.

[50] S. Cao *et al.*, "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, 2019.

[51] R. Shi *et al.*, "E-LSTM: Efficient inference of sparse LSTM on embedded heterogeneous system," in *56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019.

[52] G. Nan, C. Wang, W. Liu, and F. Lombardi, "DC-LSTM: Deep Compressed LSTM with Low Bit-Width and Structured Matrices," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020, pp. 1–5.

[53] Z. Chen, G. J. Blair, H. T. Blair, and J. Cong, "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*, 2020, pp. 217–222.

[54] R. Prabhakar *et al.*, "Plasticine: A reconfigurable architecture for parallel patterns," in *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[55] G. Shomron *et al.*, "SMT-SA: Simultaneous multithreading in systolic arrays," *IEEE Computer Architecture Letters*, 2019.

[56] K. Ando *et al.*, "A multithreaded CGRA for convolutional neural network processing," *Circuits and Systems*, 2017.

[57] E. Nurvitadhi *et al.*, "Automatic multithreaded pipeline synthesis from transactional datapath specifications," in *Design Automation Conference*. IEEE, 2010, pp. 314–319.

[58] E. Baek *et al.*, "A Multi-Neural Network Acceleration Architecture," in *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020.