Recurrent Neural Networks with Column-wise Matrix-Vector Multiplication on FPGAs

Zhiqiang Que, *Member, IEEE,* Hiroki Nakahara, *Member, IEEE,* Eriko Nurvitadhi, *Member, IEEE,* Andrew Boutros, *Member, IEEE,* Hongxiang Fan, *Student Member, IEEE,* Chenglong Zeng, Jiuxi Meng, *Student Member, IEEE,* Kuen Hung Tsoi, Xinyu Niu, *Member, IEEE,* and Wayne Luk, *Fellow, IEEE*

***Abstract*—This paper presents a reconfigurable accelerator for REcurrent Neural networks with fine-grained cOlumn-Wise matrix-vector multiplicatioN (RENOWN). We propose a novel latency-hiding architecture for RNN acceleration using column-wise matrix-vector multiplication instead of the state-of-the-art row-wise operation. This hardware architecture can eliminate data dependencies to improve the throughput of RNN inference systems. Besides, we introduce a configurable checkerboard tiling strategy which allows large weight matrices, while incorporating various configurations of element-based parallelism and vector-based parallelism. These optimizations improve the exploitation of parallelism to increase hardware utilization and enhance system throughput. Evaluation results show that our design can achieve over 29.6 TOPS which would be among the highest for FPGA-based RNN designs. Compared to state-of-the-art accelerators on FPGAs, our design achieves 3.7 to 14.8 times better performance and has the highest hardware utilization.**

I. Introduction

RECURRENT Neural Networks (RNNs) have shown remarkable successes in sequence-to-sequence processing applications such as natural language processing (NLP) [1], speech-to-text [2] and video analysis [3]. Since low-latency is key for a seamless user experience in such applications, efficient and real-time RNN acceleration is required. There are many RNN variants. Long Short-Term Memory (LSTM) and Gated recurrent unit (GRU) are the two most popular ones. To speed up RNN inferences, FPGAs have been utilized in various scenarios [4]–[7], achieving lower latency and power consumption compared to CPUs and GPUs.

However, the recurrent nature and data dependency in the RNN computation results in undesired system stall until the required hidden vectors return from the full pipeline to start the next time-step calculation [6]. Besides, while deep pipelining can be utilized to enhance operating frequency, it increases stall penalty due to longer drain time. Moreover, inefficient tiling can leave hardware resources idle resulting in low utilization. For example, the Brainwave [6] has six matrix-vector multiplication (MVM) "tile engines", each processing 400x40 matrices, so they have a peak capability of processing a 400x240 matrix in parallel. Any MVM that does not map to this dimension will leave some resources idle. Fig. 1 shows that Brainwave's hardware utilization ranges from less than

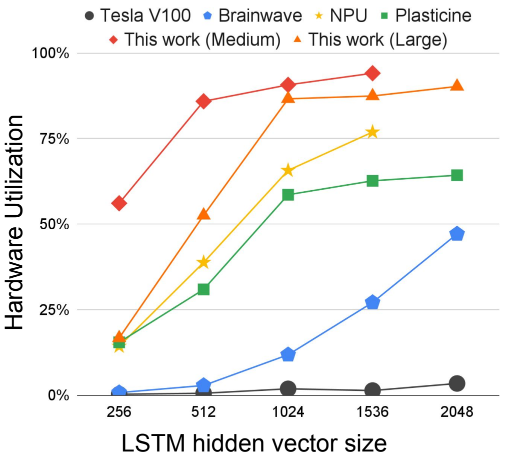


Fig. 1. Hardware utilization of various LSTM implementations.

1% to only about 50% for various LSTM models. Another implementation, Google's TPU, also suffers from low hardware utilization and achieves an average utilization of 3.5% for LSTMs [8]. The Brainwave-like Neural Processor Unit (NPU) [7] with a fine-grained zero-padding scheme achieves around 75% for a large LSTM. However, it still suffers from low utilization, especially from medium to small sized LSTMs which are commonly used in many applications [3], [4], [9]. LSTM models with small to medium sizes that have a 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 [10]. With the need for high-performance systems, it is essential to maximize the hardware utilization to achieve the highest possible effective performance and energy-efficiency.

This paper proposes a novel latency-hiding hardware architecture and a configurable checkerboard tiling strategy for RNN/LSTM models to increase hardware utilization and enhance the throughput of RNN inference. First, we propose column-wise MVM for RNN/LSTM gates, which is able to eliminate their data dependencies. The column-wise block-striped decomposition of a matrix in MVM, as shown in Fig. 2b, is an effective outer-product based parallel method for processing MVM in high-performance computing. Recently there are also some outer-product based matrix-matrix multiplication accelerators [11], [12]. However, most of the previous FPGA-based RNN implementations focus on row-wise MVM as shown in Fig. 2a. The proposed architecture can start the calculation of the next time-step without waiting for the system pipeline to be drained, which means that the system can be fully pipelined without stalling.

Z. Que, H. Fan, J. Meng and W. Luk are with Imperial College London, UK. E-mail:{z.que, h.fan17, jiuxi.meng16, w.luk}@imperial.ac.uk. H. Nakahara is with the Tokyo Institute of Technology, Japan. E-mail:nakahara.h.ad@m.titech.ac.jp. E. Nurvitadhi and A. Boutros are with the Intel PSG, USA. E-mail:eriko.nurvitadhi@intel.com

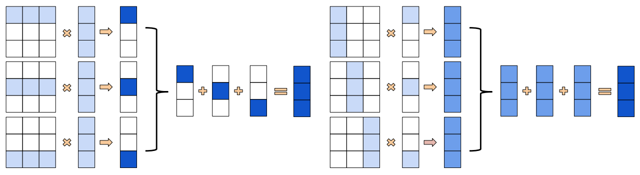

(a) Row-wise MVM (b) Column-wise MVM

Fig. 2. Matrix-vector multiplications (MVM)

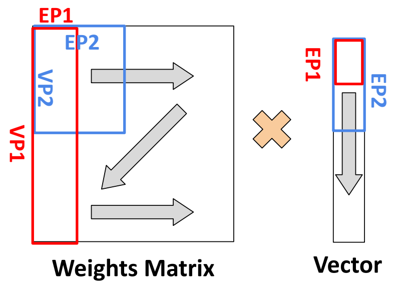


Fig. 3. The Element-based Parallelism (EP) and Vector-based Parallelism (VP) with tiles shaded in red or blue. Two sets of (EP, VP) are shown in this example.

Moreover, a novel configurable checkerboard tiling strategy is proposed which incorporates Element-based Parallelism (EP) and Vector-based Parallelism (VP) to boost inference throughput, as shown in Fig. 3. To support EP and VP, a new hardware architecture that supports hybrid kernels is proposed which combines multiplier-adder-tree and multiply-accumulate architectures. The architecture deploying many parallel multipliers followed by a large balanced adder-tree is commonly used in FPGA-based RNN/LSTM accelerators [4], [6], [7], [13]. These designs are based on row-wise MVM. To support MVM column-wise processing, our design deploys many parallel multipliers followed by accumulators, as shown in Fig. 2b. Furthermore, unlike our previous work [14] which is based on a fixed-size tiling approach, this work proposes a configurable tiling technique that supports various configurations of EP and VP to further improve the performance since different sizes of RNN models prefer different configurations of (EP, VP), as shown in Fig. 3

Our results indicate that the proposed acceleration architecture is not only the fastest compared to the state-of-the-art for a large LSTM model, but also much more suitable for a wider-range of RNN models in terms of complexity. Hence, it performs much better for RNN models with different sizes as shown in Fig. 1. We make the following contributions:

- Novel column-wise MVMs for RNNs to eliminate data dependencies, increasing the hardware utilization and system throughput.
- A flexible checkerboard tiling strategy supporting EP and VP to exploit the available parallelism while increasing hardware utilization and scalability. Besides, the (EP, VP) parameter space is comprehensively explored.
- A latency-hiding hardware architecture with novel hybrid kernels and Configurable Adder-tree Tail (CAT) units to support the proposed optimizations.
- Experimental evaluation of the proposed approach and hardware architecture, showing performance improvement of 3.7 to 14.8 times over state-of-the-art FPGA-based LSTM designs [6], [7], [15], with the highest hardware utilization among them.

Relationship to Prior Publication: This paper extends our previous work in [14], which covers a medium-scale design with a fixed EP and VP configuration, limiting the exploitation of parallelism and diversity in RNN applications. Moreover, the requirements to achieve high hardware utilization for a small LSTM workload and a large LSTM workload are different. The choice of the mapping are highly dependent on the detailed layer shape, and a fixed mapping is not ideal for different models or even different layers of a model. This issue becomes severe when the number of PEs is large. An additional contribution of this paper is a runtime configurable tiling strategy which supports various sets of EP and VP, along with novel Configurable Adder-tree Tail (CAT) units, addressing the limitation described above. Besides, an extensive design space exploration is performed to identify favorable tiling block sizes for RNN models. Our approach can benefit a large-scale design involving 65,536 PEs, which has four times more PEs than our previous design [14], while still achieving high run-time hardware utilization with the same benchmarks. Furthermore, additional information about the hardware architecture and related work is included to provide a more detailed comparison. These optimizations lead to significant throughput increase and latency reduction over the previous design [14].

II. Background and Preliminaries

A. Recurrent Neural Networks (RNNs)

LSTMs, one of the variations of RNNs, were initially proposed in 1997 [16]. This work follows the standard LSTM cell [3], [6], [7], [15]. A LSTM cell is composed of four LSTM gates and an LSTM tail. Each gate has a back-to-back MVM and activation operations while the tail mainly involves vector element-wise operations. The arithmetic of a single cell iteration is listed in the following equations.

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

where symbols σ, $tanh$ and $\odot$ are respectively the sigmoid function, the hyperbolic tangent function and element-wise multiplication. i_t, f_t, g_t and o_t stand for the input, forget, input modulation and output gates at time-step t, respectively. The input modulation gate is often considered as a sub-part of the input gate. W represents the weights matrix for both input and hidden units. b represents bias. 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. Another variant of LSTMs is the Gated Recurrent Unit (GRU). The forget and input gates of the LSTM are combined into a single "update gate" in the GRU. It has fewer parameters since it has only three gates. This study focuses on the optimization techniques for the standard LSTMs, but these techniques can be generalized to other RNN variants.

B. Related Work

There has been much previous work on FPGA-based implementations of persistent LSTM whose weights are stored in on-chip memory [6], [7], [13], [17]–[19]. There are also some previous studies of LSTM implementations storing weights in off-chip memory on FPGA devices [4], [20]–[23], which had been identified as the performance bottleneck. [24] utilizes stochastic computing to improve the energy efficiency of RNNs. [25] proposes reversible logic gates for low power circuit designs for LSTMs. BLINK [26] utilizes bit-sparse data representation for LSTMs. POLAR [27] and BRDS [28] present FPGA-based pipelined and overlapped architecture for dense and sparse LSTM inferences respectively. [29] proposes a multi-FPGA based approach for accelerating deep RNNs. [30] explores RNN partitioning strategies to achieve scalable multi-FPGA acceleration for large RNNs. Some of the previous studies [2], [31]–[36] are focusing on weight pruning and model compression to achieve good performance and efficiency. [37]–[40] present block circulant matrix to reduce LSTM inference weight matrices. These studies are orthogonal to our proposed strategy and architecture, and can be complementary to our approach to achieve even higher throughput of RNN inferences on FPGAs.

The authors in [15] propose the cross-kernel optimization within RNN cells targeting Plasticine, which achieves 30 times higher performance compared to a GPU. The Brainwave [6] is a single-threaded SIMD architecture for persistent RNNs. It achieves more than an order of magnitude improvement in latency and throughput over GPUs. [7] introduces a Brainwave-like neural processing unit (NPU) for RNNs. A TensorRAM is also proposed for large persistent data intensive RNN models. Related work [19], [41] proposes a novel hardware architecture of 2D-LSTM and investigates the trade-off between precision and performance. The authors in [42] present the first performance evaluation of Intel's new AI-optimized FPGA, the Stratix 10 NX, with AI tensor blocks. All these RNN designs are based on row-wise MVM and suffer from data dependencies. Deploying the proposed latency-hiding hardware architecture involving column-wise MVM and the proposed tiling strategy, RENOWN achieves high hardware utilization and throughput.

III. Design and Optimization

This section covers data dependency analysis and optimizations targeting RNN designs. Some system parameters are defined in Table I.

A. Weights Matrix of LSTM Gates

In this design, the four matrices of i, f, o, u gates in LSTMs are combined into one large matrix since they share the same size. Thus, in one time-step calculation of the LSTM, we only need to focus on 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 MVMs that share the same input vector. Since each gate matrix has the size of $Lh \times (Lx + Lh)$, the size of the combined matrix is $(4 \times Lh) \times (Lx + Lh)$. Then we have the $Hw = 4 \times Lh$ and $Lw = Lh + Lx$. Besides, the weights of the four LSTM gates are also interleaved in the final weights matrix. Therefore, 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 units.

TABLE I
Summary of parameters used in our study

W	Weights matrix
w_n	All the weights of row n in W
w'_n	All the weights of column n in W
Hw	Number of Rows of weights matrix
Lw	Number of columns of weights matrix
x_t	The input vector x at timestep t
h_t	The hidden vector h at timestep t
$x_t[j]$	The element j in the input vector x at timestep t
$h_t[j]$	The element j in the hidden vector h at timestep t
Lx	Number of elements in the input vector x
Lh	Number of elements in the hidden vector h
NPE	Number of processing elements
EP	Element-based Parallelism
VP	Vector-based Parallelism
TS	Timestep
MVM_x	The MVM involving input vector x
MVM_h	The MVM involving hidden vector h

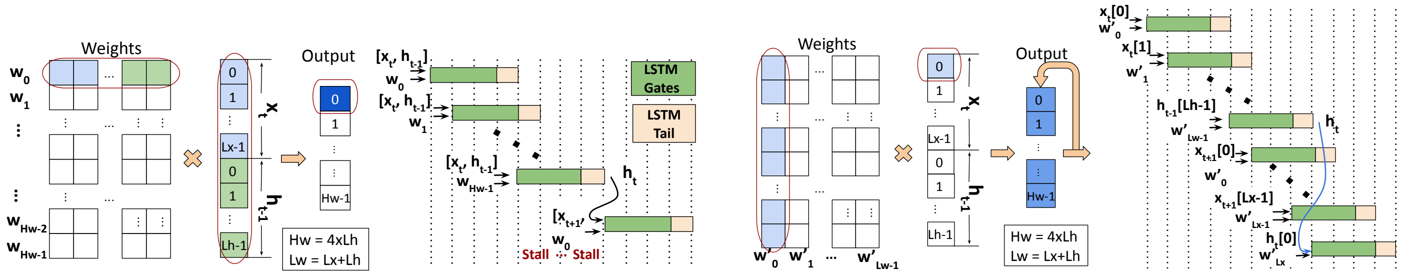
B. Row-wise MVM for RNNs

Conventional designs of MVM for RNNs are row-wise, and they have a major problem of being stalled when their pipelines are not fast enough to bring data back to the input for the next time step. They involve the entire vector of (x_t, h_{t-1}) and one or several entire rows of the weights matrix at a time, as shown in Fig. 4a. This approach imposes additional stalling since the system has to wait for a newly computed hidden vector before starting the calculation of next timestep.

Data hazard exists since the whole new hidden vector h_t is required to start the new computation of x_{t+1} in the conventional MVM design for RNN/LSTM. It is mainly due to the data dependencies between the output from the current timestep and the vector for the next timestep. It indicates that the whole system pipeline needs to be drained to get the new computed hidden vector h_t before the new matrix-vector operations can start. As [6] 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 to start the next timestep then the MVM unit will stall, as shown in Fig. 4b. Therefore, pipeline latency is important. On the other hand, deep pipelining is often required to achieve a high operating frequency for designs. This makes it difficult to achieve a design with the best trade-off.

C. The Proposed column-wise MVM for RNNs

This work proposes a new technique that can alleviate this problem by calculating the matrix-vector operations in



(a) Row-wise MVM for LSTMs (b) Pipeline analysis of (a) (c) Column-wise MVM for LSTMs (d) Pipeline analysis of (c)

Fig. 4. The row-wise and column-wise MVM with LSTM data dependencies analysis

a column-wise fashion. At the beginning, only a few elements from the x_t vector are used while h_{t-1} is not touched, but all the elements in the corresponding columns of the weights matrix are involved to perform the operations, as illustrated in Fig. 4c. To illustrate the idea, the number of the pipeline stages of the example system is 4 as shown in Fig. 4d. However, the pipeline stages of a real system can be much larger. In addition, the figure shows only one element in the x_t vector is used to perform the calculation for simplicity, but the actual number of the involved elements in each cycle depends on the architecture's parallelism requirements. The number of elements employed in this work is EP which is explored and fine-tuned in Section IV. The partial result vector is generated from the small dot-product of the partial x_t vector and the corresponding weights. Then it is accumulated over multiple cycles to generate the final result vector. This way, the calculation of the new inference of (x_{t+1}, h_t) can start without waiting for the system pipeline to be drained to get h_t since it only needs a partial input vector. It indicates that the system can be fully pipelined without stalls, as shown in Fig. 4d. Each hidden vector can finish the calculation in the shadow region of processing x_t before it is required. The stall happens in the calculation of each timestep, and the total potential stalling cycles equals the design pipeline stages. When the LSTM workload is large, e.g., with a layer Lh as 2048, the number of processing cycles of such a workload will be much larger than the number of stall cycles and then the benefit of the column-based MVM will be small, because the ratio of the stall cycles to the processing cycles is small. However, when the workload is small, e.g., with a layer Lh as 256, the number of processing cycles of such a workload is also small. In such a scenario, reducing stall cycles is vital because the ratio of stalls is large. For example, if the number of processing cycles is the same as to the design pipeline stages, the hardware utilization can be increased from 50% to 100% if all the stalls can be hidden.

One disadvantage has been observed that although the column-wise MVM only needs a partial input vector, it produces the output vector later than the row-wise MVM, because it needs to wait for all the columns to be processed to get the final accumulated vector before producing the output [43]. It seems that the succeeding hardware units that depend on the output vector (e.g., those that perform activation functions and element-wise operations in the RNNs) would need to wait longer. In contrast, the row-wise MVM completes one subset of the output vector before moving to the next subset. Therefore, a subset of the final output vector is completed sooner than in a column-wise case. However, in the column-wise case, the succeeding units can get an entire output vector but not a subset. Although the column-wise architecture starts the subsequent processing later than the one using row-wise MVM, the number of succeeding units can be increased to match the output bandwidth and finish the whole calculation sooner than the row-wise case. Practically, we do not need to significantly increase the number of these units since they can process the whole vector over multiple cycles, until the next vector is produced by the MVM engine. Besides, the row-based approach may also involve multiple succeeding units to increase parallelism. Moreover, these units are much smaller compared to the MVM kernel engine that has lots of multipliers and adders. Thus, the extra hardware area is negligible compared to the whole accelerator.

When the size of x_t vector is small and/or the size of h_t vector is large, the design may still stall because the cycles of processing x_t vector cannot completely hide the whole pipeline latency to get the h_t ready before it is needed. However, with the column-wise MVM, we can still continue to process the MVM of x_t and its corresponding weights when we are waiting for the h_t to be computed. Besides, when the input vector is small, an LSTM model would rarely require a significantly larger hidden vector.

D. Tiling and Parallelism

To further exploit the available parallelism, we introduce Element-based Parallelism (EP) and Vector-based Parallelism (VP) in our design, as shown in Fig. 3. The matrix of weights is split into small tiles with a size of (EP, VP). In each cycle, the hardware engine is able to process a tile of the weights matrix and a sub-vector of $[x_t, h_{t-1}]$ with a size of EP. EP and VP need to be determined carefully so that the number of cycles to process the x_t vector, given by $\frac{Lx}{EP}$, is larger than the system latency to ensure that the computation of hidden vectors can be fully hidden by processing x_t vector. This number is small when EP is large and it may still result in system stalls. 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 4 gates in LSTM. In summary, the hardware utilization and system throughput can be improved via balancing EP and VP.

A fixed configuration of (EP, VP) can bring low hardware utilization. Brainwave [6] has 6 MVM tile engines, each

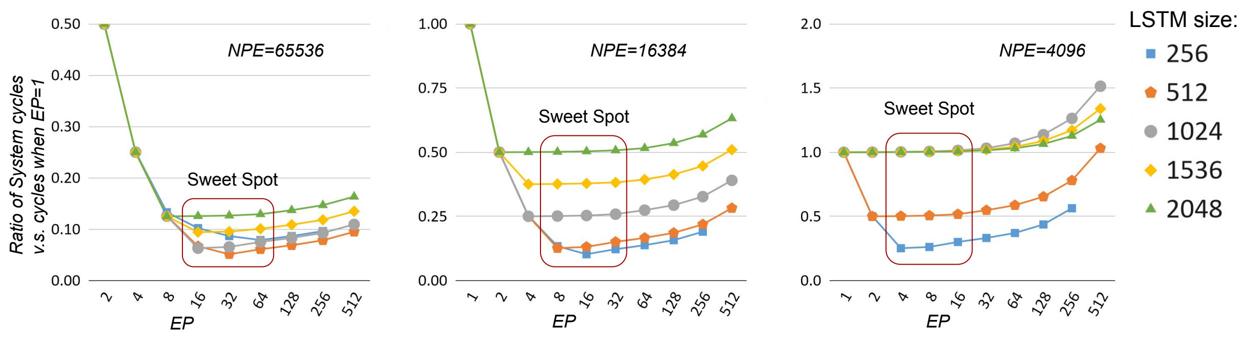


Fig. 5. The number of processing cycles in our proposed column-wise approach for different values of EP, NPE, and model sizes. Different colors combining different point shapes represent different model sizes with Lh from 256 to 2048.

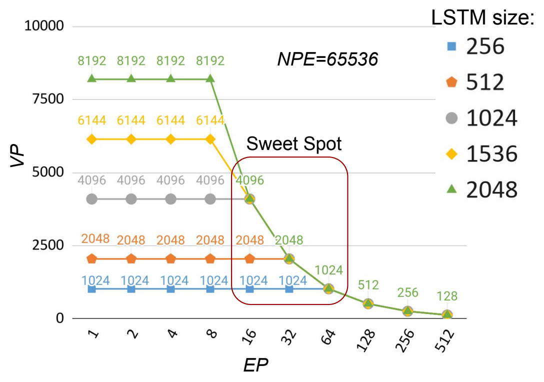


Fig. 6. Feasible sets of (VP, EP) when NPE equals 65536. The Sweet Spot is set according to the sweet spot in Fig. 5(left).

processing 400×40 matrices. The NPU [7] also has a fixed configuration of 4 tiles, 120-wide dot product engines and 40 lanes. Any MVMs that do not map well to these dimensions will leave some resources idle. Since RNNs are used for various tasks, RNN accelerators should support diverse configurations. This work proposes a novel hardware architecture to support various sets of (EP, VP) since models with different sizes may prefer different optimal EP and VP configurations. Fig. 3 shows the basic idea with two sets of (EP, VP).

One option is to adopt the row-wise MVM while cascading the computation of MVM_x and MVM_h, which are the MVMs involving input vector and hidden vector respectively, instead of a unified MVM. Cascading them with a row-wise fashion may also help to eliminate the data dependencies between current and next timestep calculations. However, it brings many issues. First, it introduces new data dependencies. Since the partial results from the MVM_x and MVM_h need to be added together to generate the final output, if we calculate the MVM_x first, we have to store the partial results somewhere locally waiting for the partial results from MVM_h operations. This introduces a memory overhead of size $4Lh$ (for LSTMs). When Lh is large, this overhead is large because the design not only needs to cache these partial results but also has to fetch them to accumulate them with the corresponding partial results from the hidden vector part. In our design, only the VP partial results are needed which means only VP registers are needed. Besides, these partial results are available in the next cycle without prefetching so they can be accumulated easily. Second, it brings difficulty in enhancing parallelism. Usually the length of x and h are different, resulting in different computation loads for MVM_x and MVM_h. The input vector x is usually application dependent while the value of h can be selected by designers to meet application requirements and to reduce hardware utilization. Zero padding may be required to support both MVM_x and MVM_h, which causes inefficiency. Actually the height of the MVM_x and MVM_h are both $4Lh$. The column-wise version of this computation has more parallelism than the row-wise version, which improves design efficiency. Both the designs [6], [7] separate and cascade the MVM_x and MVM_h, but they still suffer from low hardware utilization as explained above.

IV. Design Space Exploration

The hardware design space is characterized by the tiling block size of (EP, VP) and the number of processing elements (NPE) after combining the configurations discussed previously. The effective performance varies with the tile size and the number of PEs. To figure out the optimal parameters of the system configuration for our in-depth analysis, we develop a cycle-accurate simulator to conduct the design space exploration. A greedy algorithm is proposed to explore design space. It starts with $EP = 1$ while the VP is given according to the system constraints shown in equations (1).

$$VP \leq 4 \times Lh \quad \text{and} \quad VP \leq \frac{NPE}{EP} \tag{1}$$

Practically EP and VP should be as large as possible because this would maximize the potential parallelism and the system throughput. However, when EP increases, the number of cycles needed for processing the input vector (Lx) decreases so that the system may not have sufficient cycles to completely hide the processing of the hidden vector as discussed in Section III-C. In this exploration, the VP is set as large as possible, which is $min(4 \times Lh, \frac{NPE}{EP})$. Fig. 5 illustrates the exploration results for different sizes of LSTM models with Lh from 256 to 2048 with different colors and point shapes, using our hardware design when the NPE is 4096, 16384 or 65536. Fig. 6 shows the feasible sets of (EP, VP) when NPE is 65536. The number of processing cycles determines the throughput of the system and the lower it is, the better the overall performance will be. As shown in Fig. 5, when EP is small, the number of processing cycles is high because the VP is constrained by Equation (1) so that the number of effective PEs is less than NPE, which leads to severe underutilization.

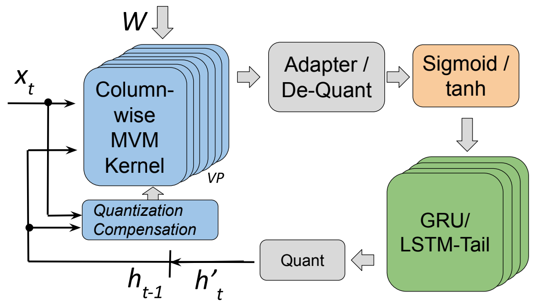


Fig. 7. The overview of the system

For instance, VP should be no larger than $4 \times Lh$ which is 1024 when targeting the LSTMs with the size of Lh being 256, illustrated by the blue line in Fig. 6. When EP increases, the number of processing cycles decreases until EP reaches these sweet spots. When EP is larger than the ones in sweet spots, processing cycles increase gradually. Please note $EP = 1$ is not shown in Fig. 5(left) as its value is too large and it will make the sweet spot too small to be shown.

From our design space exploration result, the optimal configuration has an EP value between 4 and 16 when $NPE = 16,382$, and between 16 and 64 when $NPE = 65,536$. In these sweet spots, high parallelism can be achieved, which results in high system throughput. Note that for a given vector size, better performance and utilization can be obtained by adapting the (EP, VP) design parameters. As shown in Fig. 5 (left), the LSTM workload of 512 has lower latency with (EP, VP) of (32, 2048) than that of (16, 4096) as shown by the red line, while the workload of 1024 has lower latency with (16, 4096) than (32, 2048) as shown by the gray line. Different choices of EP and VP impact the hardware utilization and performance of the architecture when running RNN models of different sizes. There is a trade-off between performance and design complexity with extra hardware resources for supporting various values of (EP, VP). More details about this trade-off is given in the following section.

V. Hardware Implementation and Optimization

This section presents our proposed hardware architecture (Fig. 7), based on the optimization techniques introduced above. It consists of the kernel units, an adapter unit, an activation function unit and tail units.

A. Kernel Units

The architecture consists of VP kernel units, and each unit has EP Processing Elements (PEs), so the number of effective PEs is $VP \times EP$. The VP and EP values are determined via the design space exploration described in detail in Section IV. In this design, each PE is one fully-pipelined multiplier. Fig. 8c shows the details of a computational kernel unit in our design. The architecture of kernel units, as shown in Fig. 8a, which employs many parallel multipliers followed by an balanced adder-tree is commonplace in FPGA-based designs of RNNs [4], [6]. This architecture is for row-wise MVMs. The column-wise MVM is based on the architecture of many parallel multipliers followed by many parallel accumulators, as shown in Fig. 8b, since the elements in the

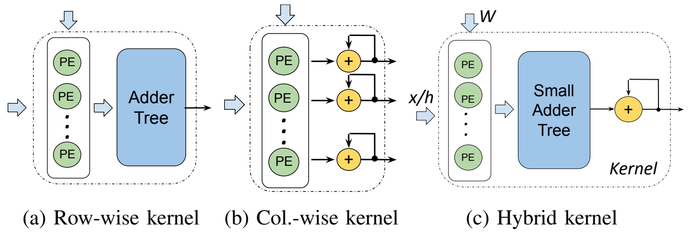


Fig. 8. Various Kernels

partial result vector are not related. To support element-based parallelism, we propose a hybrid hardware architecture that combines these two architectures. A small balanced adder tree is placed between the multipliers and the accumulators, as shown in Fig. 8c. This small adder tree, which provides the summation of the products of EP multiplications, can help to balance the EP and VP for a proper shape of a tile. For example, when the VP is limited by $4 \times Lh$ as shown in Fig. 6, the design can increase the EP to enable more PEs to increase the throughput since the number of effective PEs is $VP \times EP$.

The hybrid kernel might look more complex than the row-wise or column-wise kernel but it does not consume more hardware resources when targeting a same problem size. For example, if the number of the multipliers is N (for simplicity, N is a power of 2) for all 3 types of kernels and each hybrid kernel has a EP-to-1 small adder-tree, the design with row-wise kernels requires N multipliers and $N-1$ adders (the design is supposed to be a fully pipelined one with a balanced tree), the design with column-wise kernels requires N multipliers and N accumulators, and the design with hybrid kernels needs N multipliers, $\frac{N}{EP} \times (EP-1)$ adders and $\frac{N}{EP}$ accumulators. Because one EP-to-1 balanced adder-tree unit needs $EP-1$ adders and there are $\frac{N}{EP}$ of such units, so there are $\frac{N}{EP} \times (EP-1)$ adders in total. Generally, an accumulator is just an adder so the hybrid kernel also needs $\frac{N}{EP} \times (EP-1) + \frac{N}{EP} = N$ adders. Thus the design with hybrid kernels has the same amount of hardware resources as the one using pure column-wise kernels and just one more adder than the one using row-wise kernels. Please note that some row-wise architectures also have accumulators after the large reduction tree since it can never guarantee to fully unroll the matrix dimension [7]. In such a case, the design using row-wise kernels also requires N adders.

B. Configurable Adder-tree Tail (CAT)

To support various versions of EP and VP, we design novel adder reduction based on a custom adder-tree with the Configurable Adder-tree Tail (CAT). With various EPs, the number of levels of the adder-tree needs to be changed correspondingly. If a fixed structure of the adder-tree is designed for a large EP (EP is also the number of the input elements for the adder-tree), the results from the last several levels of adder-tree can be used for small EPs to update the accumulators directly instead of entering next level adders, as shown in mode 2 and 3 in Fig. 9. For example, if EP is 64 and the number of input is also 64, with the proposed 4-input CAT, the number of the output elements of this adder-tree becomes

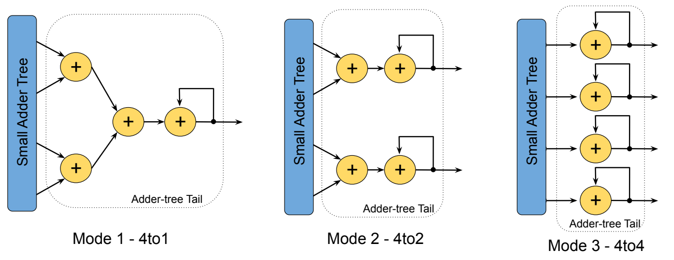


Fig. 9. The three modes of configurable 4-input adder-tree tail with accumulators

2 when mode 2 is enabled. Thus, instead of enabling a 64-to-1 adder-tree reduction, the design now has a 64-to-2 adder-tree which actually includes two 32-to-1 adder-trees. So the configuration of (EP, VP) changes from (64, 1024) to (32, 2048). With mode 3, the same design can be enabled with (16, 4096). The detailed implementation can be achieved by additional accumulators while keeping the tree structure intact. However, the additional accumulators will result in resource overhead. This work proposes the CAT architecture to reuse the adders in the tail of the tree as the required accumulators with no extra adder components. The CAT with N-input (CAT-N) can be configured to update 1 to N accumulators when the data reach the last $log_2(N)$ levels of the adder-tree. Fig. 9 shows the three modes using CAT-4. The results from the adder-tree can be used to update 1 to 4 accumulators. Fig. 10 shows the details of a CAT-4 unit. Different MUX settings configure CAT-4 to be one of the 3 modes shown in Fig. 9. For example, the red line in Fig. 10 shows the data flow when CAT-4 in mode 2. In our large scale design, CAT-4 is sufficient since the sweet spot of EP is $\{16, 32, 64\}$ according to design exploration for optimal system throughput.

C. The Other Units

The adapter converts the parallelism between kernels and tails. Then de-quantization (De-Quant) converts quantized values into fixed-point values to reduce hardware resources. The σ/tanh unit performs the Sigmoid (σ) and hyperbolic tangent ($tanh$) functions. Both target programmable lookup tables of size 2048 [2], [7]. The LSTM-tail unit and GRU-tail unit mainly perform the element-wise operations. The output hidden vector (h_t) needs to be quantized before it can be used in the MVM kernels, so a Quant unit is deployed after the final output of Tail units as shown in Fig. 7.

D. Low Precision Multiplications with DSP block Sharing

Reducing the precision of operations in DNN inference accelerators can achieve high efficiency with little or no accuracy loss compared to floating-point by fitting more multipliers per unit area. With careful retraining, low precision, even binarized RNNs can still have decent accuracy [19], [44], [45]. The authors in [19] trained an LSTM model using 1-bit weights and 2-bit activations, which achieved a classification accuracy of 94% for OCR applications. Besides, narrow bit-width multiplications can be mapped efficiently onto lookup tables and DSPs. For example, Brainwave [6] deploys 96,000 MACs on a Stratix 10 2800 FPGA by packing 2-bit or 3-bit

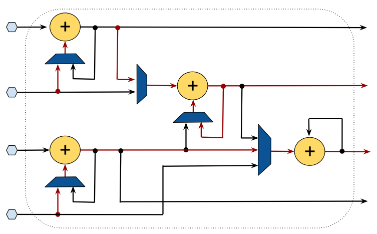


Fig. 10. The details of CAT-4

multiplications into DSP blocks combined with cell-optimized soft logic multipliers and adders. Our fixed 8-bit design has 16,384 MACs. And our fixed 2-bit design has 65,536 MACs, which deploys the same word length for multipliers as those in Brainwave targeting the same FPGA device for a fair comparison. The resource consumption of 65,535 fixed 8-bit multipliers exceeds the available resources in the Stratix 10 2800 FPGA. Our column-wise MVM optimizations and fine-grained tiling strategy are applicable to multipliers of any numerical precision.

In current FPGAs, there are highly configurable DSP blocks which are often underutilized in implementing low precision DNN designs. [46] and [47] demonstrated methods to pack two 8-bit multiplications into one Xilinx and Intel 18-bit multiplier respectively. Both methods require two multiplications to share one input operand. With the proposed column-wise MVM, one column of the weights matrix naturally shares the same element of the input vector, which helps us to pack four 8-bit or ten 2-bit multiplications into one DSP block on Intel FPGAs [47] to reduce the hardware resources. Moreover, this would not be a restriction (and will come at lower cost) if we use a novel DSP similar to what was proposed in [48] and will be adopted in the next generation Agilex devices [49].

VI. Evaluation and Analysis

This section presents evaluation results and analysis of two generations of Intel FPGAs to show the scalability of the proposed RNN accelerator optimizations.

A. Experimental Setup

To evaluate our RNN design, we utilize the same LSTM and GRU workloads as the Brainwave design [6], the Brainwave-like NPU [7] and the Plasticine [15] for comparison. These workloads come from the DeepBench suite which is a set of micro-benchmarks containing representative layers from popular DNN models such as DeepSpeech [50]. These workloads are single layers with various sizes of hidden vectors and different timesteps (or sequence lengths) from 1 to 1500, as shown in Table III. This work takes the LSTMs with Lh=256 as small LSTM workloads, Lh=512 or 1024 as medium ones, and Lh=1536 or 2048 as large ones. Two generations of Intel FPGAs, an Arria 10 1150 (A10) and Stratix 10 2800 (S10) are evaluated and compared with previous work. Both run persistent LSTM/GRU of inference. Our proposed hardware architecture is implemented in Verilog hardware description language, and implemented on the target A10 and S10 devices using Quartus Pro 18.1.

B. Resource Utilization

Table II shows the resource utilization of our designs with three configurations on FPGAs. We implement a small RENOWN with the configuration of (EP, VP) as (4, 1024) using an Arria 10 FPGA which has 4096 8-bit multipliers in the MVM kernels. A medium RENOWN with the configuration of (EP, VP) as (16, 1024) is implemented using a Stratix 10 FPGA which includes 16,384 8-bit multipliers. A large RENOWN with 65,536 2-bit multipliers is also implemented on a Stratix 10 FPGA with the configurations of (EP, VP) as (16, 4096), (32, 2048) and (64, 1024). All our designs consume most of the FPGAs' available resources. Note that hardware utilization is different from resource utilization and it reflects how often the hardware computational units would be "not idle". Although we achieve a similar frequency to that reported in the Brainwave [6] and Intel-NPU [7] papers, we believe that further low-level optimizations can lead to higher frequencies for better performance. We leave that for future work since it has a limited impact on the conclusions in this paper.

C. Performance and Efficiency Comparison

To illustrate the benefits of our proposed approach, some existing LSTM/GRU accelerator designs using the same benchmark are compared with ours in Table III. This table illustrates the latency, hardware (HW) utilization and throughput with various workloads under different numbers of hidden units (h) and time-step (TS). The hardware utilization is the percentage of achieved Tera Operations Per Second (TOPS) to the peak performance for each layer. The DeepBench published results [15] on a modern NVIDIA Tesla V100 GPU with 16-bit precision are also included. Some existing FPGA-based LSTM accelerator designs are listed in Table IV. For a fair comparison, we only show the previous work with a detailed implementation of the LSTM system in this table. We show the FPGA chips, model storage, precision, number of processing elements (NPE), run-time frequency, throughput, power efficiency and hardware utilization.

The GPU is significantly underutilized even when cuDNN library API calls, since it is designed for throughput-oriented workloads, and it prefers BLAS level-3 (matrix-matrix) operations which are not common in RNN workloads [15]. Our design can provide promising latency under 3ms for all Deepbench RNN layers at batch size of one, reaching up to 29.6 effective TOPS for a large LSTM workload (h=2048) which is the largest reported performance in all these LSTM designs as shown in Table III and Table IV. Our work achieves 27.4 to 95.8 times higher performance than the Tesla V100 as shown in Fig. 11a. The performance of the specific case (h=512) is an approximate two orders of magnitude advantage over the Tesla V100.

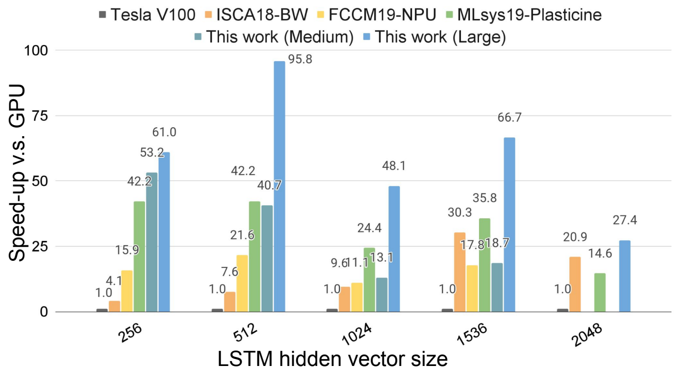
Our experiments show that the utilization is low with small RNN applications that are composed of sequences of small MVMs due to small hidden unit sizes and large number of time-steps. However, with our proposed optimizations, we can get higher throughput and hardware utilization than the counterparts using a similar number of PEs. With a similar number of PEs to [7], our RENOWN (Medium) achieves up to 94.1% hardware utilization which is the highest with respect to state-of-the-art implementations on FPGAs, as shown in Fig. 1 and Fig. 11b. Achieving high utilization using a small number of PEs is easier than using a large number of PEs. In our RENOWN (Large) design, we use around 31.7% fewer multipliers than [6], while achieving 30.7% higher performance than [6] when targeting large LSTM workloads. When targeting small LSTM models, our design achieves 14.8 times higher performance. This is due to better hardware utilization which comes from our optimizations incorporating novel column-wise MVM and fine-grained tiling strategy. With a large number of PEs, our RENOWN can still achieve up to 90.2% hardware utilization which is higher than the other work. When targeting small LSTMs and GRUs, RENOWN

TABLE II
RESOURCE UTILIZATION

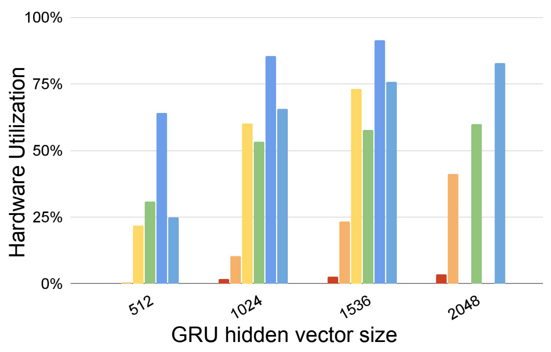
		ALMs	M20K	DSP	Freq.
Arria 10 1150 (Precision = 8-bit, NPE = 4096)	Avail.	427,200	2713	1518	259Mhz
	Used	186,534	1,178	1,176	
	R. Util.	44%	43%	77%	
Stratix 10 2800 (Precision = 8-bit, NPE = 16,384)	Avail.	933,120	11,721	5760	260Mhz
	Used	487,232	10,061	4,368	
	R. Util.	52%	86%	76%	
Stratix 10 2800 (Precision = 2-bit, NPE = 65,536)	Avail.	933,120	11,721	5760	250Mhz
	Used	768,406	6,803	5,368	
	R. Util.	82%	58%	93%	

TABLE III
PERFORMANCE COMPARISON OF DEEPBENCH INFERENCE FOR THE PREVIOUS WORK AND OUR DESIGNS

Benchmark		GPU	[6]	[7]	[15]	This Work (Medium)	This Work (Large)
NPE		-	96000	19200	12288	16384	65536
GRU h=512 TS=1	latn. (ms)	0.39	0.013	0.0015	0.0004	0.0006	0.0004
	HW Util.	0.03%	0.5%	21.7%	30.9%	64.1%	24.8%
	Perf. (TOPS)	0.01	0.25	2.17	7.6	5.46	8.13
GRU h=1024 TS=1500	latn. (ms)	33.77	3.792	3.139	1.4430	2.59	0.879
	HW Util.	1.8%	10.4%	60.2%	53.3%	85.5%	65.5%
	Perf. (TOPS)	0.56	4.98	6.01	13.1	7.28	21.5
GRU h=1536 TS=375	latn. (ms)	13.12	0.951	1.454	0.7463	1.36	0.428
	HW Util.	2.6%	23.3%	73.2%	57.8%	91.4%	75.8%
	Perf. (TOPS)	0.81	11.17	7.30	14.2	7.79	24.8
GRU h=2048 TS=375	latn. (ms)	17.70	0.954	-	1.283	-	0.695
	HW Util.	3.4%	41.2%	-	59.81%	-	82.8%
	Perf. (TOPS)	1.07	19.79	-	14.7	-	27.1
GRU h=2560 TS=375	latn. (ms)	23.57	0.993	-	1.973	-	1.076
	HW Util.	4.0%	61.8%	-	61.0%	-	83.6%
	Perf. (TOPS)	1.25	29.69	-	15.0	-	27.4
LSTM h=256 TS=150	latn. (ms)	1.69	0.425	0.110	0.0419	0.033	0.029
	HW Util.	0.3%	0.8%	14.3%	15.5%	56.1%	16.7%
	Perf. (TOPS)	0.09	0.37	1.43	3.8	4.79	5.49
LSTM h=512 TS=25	latn. (ms)	0.60	0.077	0.027	0.0139	0.014	0.0061
	HW Util.	0.6%	2.8%	38.8%	30.9%	85.9%	52.6%
	Perf. (TOPS)	0.18	1.37	3.89	7.6	7.33	17.2
LSTM h=1024 TS=25	latn. (ms)	0.71	0.074	0.064	0.0292	0.054	0.015
	HW Util.	1.9%	2.8%	65.7%	58.6%	90.7%	86.6%
	Perf. (TOPS)	0.59	5.68	6.56	14.4	7.73	28.4
LSTM h=1536 TS=50	latn. (ms)	4.38	0.145	0.246	0.1224	0.236	0.066
	HW Util.	1.4%	27.1%	76.9%	63.7%	94.1%	87.4%
	Perf. (TOPS)	0.43	13.01	7.67	15.4	8.02	28.7
LSTM h=2048 TS=25	latn. (ms)	1.55	0.074	-	0.106	-	0.113
	HW Util.	3.4%	47.1%	-	64.3%	-	90.2%
	Perf. (TOPS)	1.08	22.62	-	15.8	-	29.6



(a) Performance speedup for the various LSTM dimensions



(b) HW utilization of various GRUs compared to prior work

Fig. 11. Performance comparison

(large) has a similar utilization as [7] and [15]. However, the number of PEs is respectively 3.41 times and 5.3 times more than those in [7] and [15]. The configurable tiling of RENOWN (Large) also results in an additional up to 27% higher system throughput as shown in Fig. 12. The figure shows the speedup of the system throughput compared to the lowest one among them with a given EP.

The previous designs of [19] and [41] in Table IV explore various low precision designs, showing their scalability to different bitwidth designs. Most of the other designs in Table IV only support one bitwidth while our work demonstrates both 8-bit and 2-bit designs which show the scalability of our architecture to cover different bit-width designs. Our implementations are produced using the Verilog templates with configurable parameters for each hardware module instance. Generally, [19] and [41] are more scalable in bitwidth since they do not use the DSP hard blocks on FPGAs for the computational kernels. The FPGA DSP block has a fixed bit-width and it needs a tailor-made wrapper for packaging several small multipliers into one DSP hardware block. Besides, [19] and [41] target a particular model which has four parallel and independent LSTM layers. These four independent LSTM layers can be scheduled in an interleave manner for a hardware engine to alleviate the issues of recurrent dependencies.

Some of the designs target smaller FPGAs [19], [41] than the Stratix 10. They have fewer logic resources and DSPs but they also have a smaller TDP (thermal design power) power consumption. We follow the convention in all related papers and use GOP/S for performance comparison. In addition, we also use GOPS/W, power efficiency, so that we are not taking

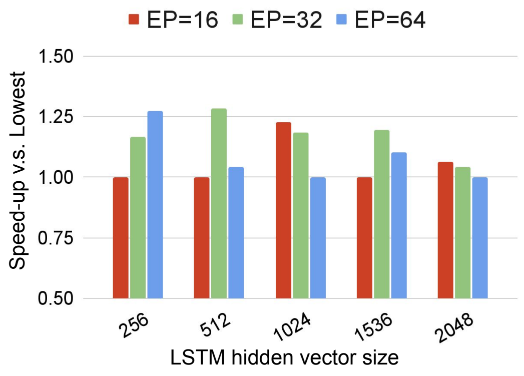


Fig. 12. Performance speedup due to configurable tiling

advantage of the FPGA chip size when compared to the designs on small chips, such as the designs in [19], [41].

Overall, our RENOWN (Medium) provides over 1.05 to 3.35 times higher performance and 1.22 to 3.92 times higher hardware utilization than the state-of-the-art design [7], as shown in Table III. In addition, the RENOWN (Large) achieves 3.7 to 14.8 times better performance than state-of-the-art FPGA-based LSTM designs [6], [7], [15]. This paper focuses on minimizing latency and maximizing throughput by increasing the hardware utilization. The results show flexible customizability of our architecture for different scenarios. The column-wise approach exposes the most parallelism while minimizing stalls due to data dependencies.

To minimize latency, our design places the model weights onto the on-chip memory, achieving high memory read bandwidth suitable for real-time services. However, some large-scale RNNs recently emerge with large on-chip memory requirements. Our design adopts a scale-out network of utilizing multiple accelerators like [6], [30] which partition the design to multiple FPGAs to address the challenges of fast-growing RNN models where weights exceed on-chip memory capacity on a single FPGA. Some cloud-based services are able to tolerate a slightly longer latency of response. It means a small amount of batching can be employed if necessary. This benefits the GPU-based RNN inferences [6]. In [13], [42], [44], [51], the batching technique is used to improve the hardware throughput and utilization for LSTM inferences. Since our design executes a single input at a time, increasing batch size does not affect the utilization. Thus, our architecture's utilization is not affected with or without batching. Furthermore, some designs use binarized datapaths [13], [19], [41] for LSTMs with negligible or no effect on accuracy. Utilizing very low precision, e.g., binary, is orthogonal to our proposed approach which transforms computation to eliminate data dependencies. Reducing precision can be combined with our approach to achieve even higher performance and efficiency.

Fig. 13 illustrates the power consumption of the proposed RENOWN (Medium) design. The power is estimated by the Intel Early Power Estimator (EPE) tool and verified by the Intel Quartus Power Analyzer. We note that this work considers only the chip power, for a fair comparison. The bar chart in Fig. 13(left) shows the decomposition for each of the major underlying FPGA components. The static power consumption of the device is 7.16W. The dynamic power consumption of

TABLE IV
COMPARISON WITH PREVIOUS IMPLEMENTATIONS OF LSTM ON FPGAS

	FPGA17 ESE [2]	FPL18- [13]	ISCA18- BW [6]	FCCM19- NPU [7]	FCCM19- NPU [7]	JSPS20- [19]	FPGA20- [41]	TVLSI20- [27]	This work (Medium)	This work (Large)
FPGA chips	Kintex KU060	Zynq ZU7EV	Stratix10 GX2800	Stratix10 GX2800	Stratix10 GX2800	Zynq ZU9EG	Zynq ZU9EG	Zynq ZU6EG	Stratix10 GX2800	Stratix10 GX2800
Model Storage	-	on-chip	on-chip	on-chip	on-chip	on-chip	on-chip	on-chip	on-chip	on-chip
Precision (bits)	12 fixed	1-8 fixed	BFP-1s5e2m	4 fixed	8 fixed	2 fixed	4/8 fixed	16 fixed	8 fixed	2 fixed
DSP Avail.	2,760	1,728	5,760	5,760	5,760	2,520	2,520	1,973	5,760	5,760
DSP Used	1,504	-	5,245	4,880	5,600	6	64	-	4,368	5,368
NPE	1024	-	19,200	96,000	38,400	-	-	-	16,384	65,536
Freq. (MHz)	200	240	250	255	260	240	240	385	260	250
Power (W)	41	-	125	65.3	67.6	15.5	15.5	0.95	62	98
Perf. (GOPS)	282	1,833	370[a] 22,620	- 14,112	1,431[a] 7,980	3,618	3,072	18	4,790[a] 8,015	5,489[a] 29,574
Power Effi. (GOPS/W)	6.87	-	180	216	118	234	199	19	129	302
LSTM HW Utilization	-	-	0.8%[a] 47.1%	-	14.3%[a] 76.9%	-	-	-	56.1%[a] 94.1%	16.7%[a] 90.2%

[a] When targeting a small LSTM model (Lh=256).

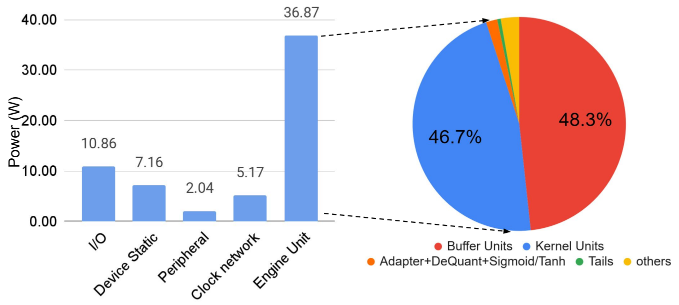


Fig. 13. Power consumption of the accelerator with the decomposition for each of the major underlying blocks.

the accelerator engine unit is the largest which is over 50%. The pie chart in Fig. 13(right) shows the dynamic power consumption of each major unit of the accelerator engine unit. The Buffer Units which store the weights and input/output data have the largest power consumption which reaches nearly half of the power consumed by the whole engine unit. The Kernel Units also consume nearly half of the power while the Tail units only consume less than 1% power.

VII. CONCLUSION AND FUTURE WORK

This paper proposes fine-grained tile-based column-wise MVMs with a novel latency-hiding hardware architecture for RNN/LSTM to improve both hardware utilization and design throughput. The proposed accelerator has been implemented using Arria 10 and Stratix 10 FPGAs, achieving superior performance and power efficiency compared to prior state-of-the-art implementations. Further research includes combining our approach with binarized RNNs, pruning techniques and in-memory computing, and automating it to support rapid development of efficient FPGA-based RNN/LSTM designs.

ACKNOWLEDGEMENTS

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

REFERENCES

[1] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, 2016.
[2] 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.
[3] 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.
[4] Y. Guan *et al.*, "FPGA-based accelerator for long short-term memory recurrent neural networks," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017.
[5] Z. Sun *et al.*, "FPGA acceleration of LSTM based on data for test flight," in *IEEE International Conference on Smart Cloud (SmartCloud)*, 2018.
[6] J. Fowers *et al.*, "A Configurable Cloud-Scale DNN Processor for Real-Time AI," in *Proceedings of the 45th Annual International Symposium on Computer Architecture*, 2018.
[7] E. Nurvitadhi *et al.*, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs," in *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 199–207.
[8] 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.
[9] W. Zaremba *et al.*, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.
[10] R. Yazdani *et al.*, "Lstm-sharp: An adaptable, energy-efficient hardware accelerator for long short-term memory," *arXiv preprint arXiv:1911.01258*, 2019.
[11] S. Pal *et al.*, "Outerspace: An outer product based sparse matrix multiplication accelerator," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.
[12] Z. Zhang *et al.*, "Sparch: Efficient architecture for sparse matrix multiplication," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.
[13] V. Rybalkin *et al.*, "FINN-L: Library extensions and design trade-off analysis for variable precision LSTM networks on FPGAs," in *28th FPL*. IEEE, 2018.
[14] Z. Que *et al.*, "Optimizing reconfigurable recurrent neural networks," in *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2020.

[15] T. Zhao *et al.*, "Serving Recurrent Neural Networks Efficiently with a Spatial Accelerator," *arXiv preprint arXiv:1909.13654*, 2019.

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

[17] J. C. Ferreira and J. Fonseca, "An FPGA implementation of a long short-term memory neural network," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig),*. IEEE, 2016.

[18] V. Rybalkin *et al.*, "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*, 2017.

[19] V. Rybalkin, C. Sudarshan, C. Weis, J. Lappas, N. Wehn, and L. Cheng, "Efficient Hardware Architectures for 1D-and MD-LSTM Networks," *Journal of Signal Processing Systems*, 2020.

[20] A. X. M. Chang *et al.*, "Recurrent neural networks hardware implementation on fpga," *arXiv preprint arXiv:1511.05552*, 2015.

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

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

[23] N. Park *et al.*, "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.

[24] Y. Liu *et al.*, "An energy-efficient and noise-tolerant recurrent neural network using stochastic computing," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, 2019.

[25] K. Khalil *et al.*, "A Reversible-Logic based Architecture for Long Short-Term Memory (LSTM) Network," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021.

[26] Z. Chen *et al.*, "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.

[27] Bank-Tavakoli *et al.*, "Polar: A pipelined/overlapped fpga-based lstm accelerator," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, 2019.

[28] S. A. Ghasemzadeh *et al.*, "BRDS: An FPGA-based LSTM Accelerator with Row-Balanced Dual-Ratio Sparsification," *arXiv preprint arXiv:2101.02667*, 2021.

[29] Y. Sun and H. Amano, "FiC-RNN: A Multi-FPGA Acceleration Framework for Deep Recurrent Neural Networks," *IEICE Transactions on Information and Systems*, 2020.

[30] D. Kwon *et al.*, "Scalable Multi-FPGA Acceleration for Large RNNs with Full Parallelism Levels," in *57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020.

[31] Z. Chen *et al.*, "CLINK: Compact LSTM inference kernel for energy efficient neurofeedback devices," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018.

[32] 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.

[33] J. Wu *et al.*, "A 3.89-gops/mw scalable recurrent neural network processor with improved efficiency on memory and computation," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, 2019.

[34] 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.

[35] 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.

[36] G. Nan *et al.*, "DC-LSTM: Deep Compressed LSTM with Low Bit-Width and Structured Matrices," in *International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2020.

[37] Z. Wang *et al.*, "Accelerating recurrent neural networks: A memory-efficient approach," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, 2017.

[38] S. Wang *et al.*, "C-LSTM: Enabling Efficient LSTM using Structured Compression Techniques on FPGAs," in *2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*.

[39] 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.

[40] K. P. Yalamarthy *et al.*, "Low-complexity distributed-arithmetic-based pipelined architecture for an LSTM network," *IEEE Trans. on Very Large Scale Integr. (VLSI) Syst.*, 2019.

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

[42] A. Boutros *et al.*, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs," in *International Conference on Field-Programmable Technology (FPT)*. IEEE, 2020.

[43] E. Nurvitadhi *et al.*, "Hardware accelerator for analytics of sparse data," in *Proceedings of the 2016 Conference on Design, Automation & Test in Europe*. EDA Consortium, 2016.

[44] A. Ardakani *et al.*, "Learning recurrent binary/ternary weights," *arXiv preprint arXiv:1809.11086*, 2018.

[45] W. Zheng *et al.*, "Binarized neural networks for language modeling," Technical Report cs224d, Stanford University, Tech. Rep., 2016.

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

[47] M. Langhammer *et al.*, "Extracting INT8 Multipliers from INT18 Multipliers," in *Field Programmable Logic and Applications (FPL)*. IEEE, 2019.

[48] A. Boutros *et al.*, "Embracing diversity: Enhanced DSP blocks for low-precision deep learning on FPGAs," in *International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2018.

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

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

[51] F. Silfa *et al.*, "E-BATCH: Energy-Efficient and High-Throughput RNN Batching," *arXiv preprint arXiv:2009.10656*, 2020.