Hardware accelerators for Recurrent Neural Networks on FPGA

Andre Xian Ming Chang, Eugenio Culurciello
Department of Electrical and Computer Engineering, Purdue University
West Lafayette, USA
Email: {amingcha,euge}@purdue.edu

***Abstract*—Recurrent Neural Networks (RNNs) have the ability to retain memory and learn from data sequences, which are fundamental for real-time applications. RNN computations offer limited data reuse, which leads to high data traffic. This translates into high off-chip memory bandwidth or large internal storage requirement to achieve high performance. Exploiting parallelism in RNN computations are bounded by this two limiting factors, among other constraints present in embedded systems. Therefore, balance between internally stored data and off-chip memory data transfer is necessary to overlap computation time with data transfer latency. In this paper, we present three hardware accelerators for RNN on Xilinx's Zynq SoC FPGA to present how to overcome challenges involved in developing RNN accelerators. Each design uses different strategies to achieve high performance and scalability. Each co-processor was tested with a character level language model. The latest design called DeepRnn, achieves up to $23\times$ better performance per power than Tegra X1 development board for this application.**

I. Introduction

Recurrent Neural Networks (RNNs) are largely used to learn from sequences of data [1], and it has been shown to be successful in various applications, such as speech recognition [2], machine translation [3] and scene analysis [4]. A combination of a Convolutional Neural Network (CNN) with a RNN can lead to fascinating results such as image caption generation [5], [6].

Long Short Term Memory (LSTM) [7] is a specific RNN architecture that implements a learned memory controller for improved training. LSTMs are largely composed of matrix-vector multiplications across multiple layers to process a single element from a sequence. In matrix-vector multiplications, only vectors are reused, because once a matrix row is processed a different matrix row needs to take place. Thus, it is hard to reuse matrix data, which is the majority of data to be processed. Adding more compute units reduces the computation time, but if memory can't supply new matrix values fast enough, then the compute units utilization rate will be low. Exploiting parallelism and hardware scaling is a challenging design task for RNNs. The key to achieve high performance in RNNs hardware accelerators is to overlap data transfer time with computation time. Continuously streaming data from off-chip memory is expensive in terms of power and it is bounded by off-chip memory bandwidth available [8]. Storing matrix values in on-chip memory is not a scalable design, since it depends on RNN model size.

The main contribution of this paper is to present different design strategies that balances memory bandwidth and internal storage utilization to optimize performance per power for RNN workloads. This paper presents three different hardware accelerators implemented on Xilinx's Zynq SoC FPGA [9]. The first architecture (DeepStream) streams all data from off-chip memory to the co-processor. It achieves high performance, but it is limited by the off-chip memory bandwidth. The second design (DeepStore) makes use of on-chip memory to store all necessary data internally. This achieves low off-chip memory bandwidth, but it is limited by the available on-chip memory. Both previous designs are either limited by off-chip memory bandwidth or available on-chip resources. The third design (DeepRnn) balances these two points to achieve high performance and scalability. As proof of concept, the hardware was tested with a character level language model made with 2 LSTM layers and 128 hidden units.

The next following sections present the background for LSTM, related work, implementation details, the experimental setup and the obtained results.

II. LSTM Background

The main feature of RNNs is that they can learn from previous information. Standard RNN can retain and use recent past information [10]. But it fails to learn long-term dependencies. This is where LSTM comes into play. LSTM is an RNN architecture that explicitly adds memory controllers to decide when to remember, forget and output. This makes the training procedure much more stable and allows the model to learn long-term dependencies [7]. The LSTM hardware module that was implemented focuses on the vanilla LSTM [11], which is characterized by the following equations:

$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_t + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i) \quad (1)$$

$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_t + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f) \quad (2)$$

$$\mathbf{o}_t = \sigma(W_{xo}\mathbf{x}_t + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o) \quad (3)$$

$$\tilde{\mathbf{c}}_t = \tanh(W_{xc}\mathbf{x}_t + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \quad (4)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (5)$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (6)$$

Where σ is the logistic sigmoid function, $\odot$ is element wise multiplication, $\mathbf{x}$ is the input vector of the layer, W is the model parameters, $\mathbf{c}$ is memory cell activation, $\tilde{\mathbf{c}}_t$ is the candidate memory cell gate, $\mathbf{h}$ is the layer output vector. The subscript $t-1$ means results from the previous time step. The $\mathbf{i}$, $\mathbf{f}$ and $\mathbf{o}$ are respectively input, forget and output gate. The

978-1-4673-6853-7/17/$31.00 ©2017 IEEE

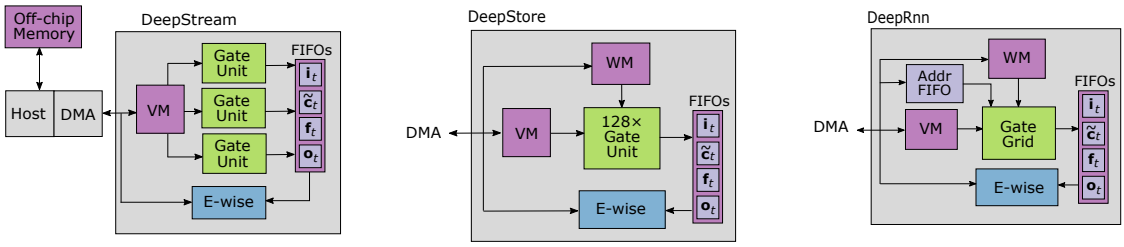

Fig. 1. Three different hardware accelerators. On the left: the DeepStream block diagram. It is mainly composed of three gate units which will process the data stream from DMA. In the middle: the DeepStore block diagram. The main difference is the weight storage unit and the number of gate units. On the right: the DeepRnn block diagram. The main difference is the weight storage is smaller and the gate units are arranged in a grid. The host processor is responsible for initiating the DMA data transfer and the configuration of the co-processor.

combination of two matrix-vector multiplications and a non-linear function, $f(W_x\mathbf{x}_t + W_h\mathbf{h}_{t-1} + \mathbf{b})$, is referred as gate. A deep LSTM network would have multiple LSTM modules cascaded in a way that the output of one layer is the input of the following layer.

III. Related Work

Co-processors for accelerating deep learning have been implemented on FPGAs [12]–[14]. Their compute engine are focused on accelerating spatial convolutions, whereas RNNs are mainly composed of matrix-vector operations, which brings different design challenges.

FPGA implementations of RNN has been explored in [15]. In [16], the RNN was unfolded into a fixed number of time-steps to be compute in parallel. Weight compression and pruning based on network sparsity are techniques that lower memory bandwidth requirement for RNNs [17]. Load balancing aware pruning is used to achieve better compute occupancy in [18].

This work presents three different implementations of RNN on FPGA. The following section presents details of our RNN accelerators.

IV. Implementation

A. Hardware

All co-processors were implemented on Xilinx Zynq-7000 SOC FPGA [9], and they are controlled by an on-chip Dual ARM Cortex-A9 MPCore. All designs use four 32 bit DMA ports, capable of achieving aggregate bandwidth up to 3.8 GB/s full-duplex. All computations are done in Q8.8 fixed point, and the accelerators run at 142 MHz.

The first design is called DeepStream and its main strategy is to stream all the data from off-chip memory to the compute units. This design is similar to the one presented in [19]. The overall design is shown in the left most diagram in figure 1.

The main building block of the implemented design is the *gate unit*, which performs the operations: matrix-vector multiplications and non-linear functions (hyperbolic tangent and logistic sigmoid). In each gate unit, there are two Multiply ACcumulate (MAC) units that compute $W_x\mathbf{x}_t$ and $W_h\mathbf{h}_{t-1}$ in parallel. The results from the MAC units are added together and it goes to an element wise non-linear module, which is implemented with piece-wise linear segmentation. The input vectors are stored in the Vector Memory (VM) until a LSTM layer is finished and new vectors come in. The intermediates results from gates units are locally stored in different FIFOs. The final result is computed by *E-wise* block that receives the data from the FIFOs and the $\mathbf{c}_{t-1}$ vector from DMA. The output vectors $\mathbf{c}_t$ and $\mathbf{h}_t$ go back to main memory through DMA. In DeepStream, 2 gate units run in parallel, because we use 4 DMA ports: 2 ports stream x_t and h_{t-1} and the other 2 stream weights. Two gate units have their non-linear module configured as sigmoid and the other gate unit is tanh. DeepStream was implemented on the Zedboard, which contains the Zynq-7000 SOC XC7Z020. The total on-chip power was 1.9 W.

The advantage of DeepStream is simplicity and high MAC utilization. The main disadvantage is the high off-chip memory bandwidth requirement, which is caused by continuously streaming of each row of each weight matrix. Another drawback is that it only uses 4 MAC units in parallel to perform the matrix-vector multiplication and scaling is limited by memory bandwidth. The second design is called DeepStore and its main strategy is to store all data into internal memory. The overall design is shown in middle of figure 1.

The main operation is the same from the DeepStream, but the difference is that all the matrix rows are stored into Weight Memory (WM) and the MAC units are replicated to multiply all matrix rows with the vector at same time. In DeepStore, the gate unit is replicated 128 times, except the non-linear function, which is shared among all MAC units. DMA is used to stream in new input vectors and to write results back to host. During initialization, DMA is used to load all weights into on-chip memory. The DeepStore architecture was implemented on the Xilinx ZC706 platform, which contains the Zynq-7000 XC7Z045 SoC. The total on-chip power is 2.3 W.

The advantage of DeepStore is that it uses low off-chip memory bandwidth and achieves high-performance. The disadvantage is that it is not scalable, because the number of MAC units and the internal memory requirement are dependent on the weight matrix height. In order to achieve scalability and high performance, we need a better arrangement of MAC units and a better data transfer strategy that doesn't hit the bandwidth or internal memory usage boundaries. A different design called DeepRnn was implemented. The design is shown on the right side of figure 1.

DeepRnn's strategy is to achieve balance between DeepStore and DeepStream designs, by storing partial weight data

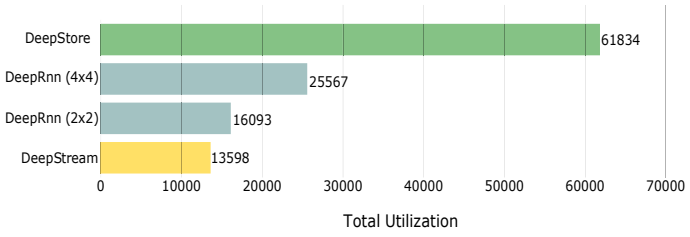

Fig. 2. FPGA utilization from each implemented design. The DeepRnn utilization scales linearly with the size of the MAC array.

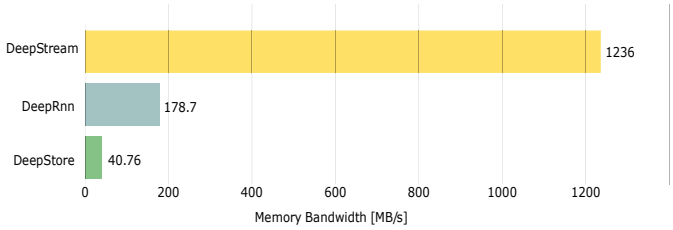

Fig. 3. Off-chip memory bandwidth requirement for each implemented architecture.

into internal memory. The main processing element of the DeepRnn is the *gate grid*, which receives data from the storage elements: Vector (VM) and Weight Memory (WM). Gate grid contains a matrix of MAC units. Each MAC unit processes a different row of the weight matrix. The vector values are the same for all MAC units. All MACs in a row are Single Instruction Multiple Data (SIMD). Different rows process and produce data independently.

VM and WM are double buffer single port SRAMs that were designed to provide data in blocks of size of number of MACs in a row. There is a WM bank for each MAC column in the grid and one VM shared for all MACs. Double buffer is the idea of having two memory banks, which are written to and read from interchangeably. This way, memory write can happen in parallel with read operation. In the case when processing time is larger than data transfer time the use of double buffer prevents the MACs from yielding, since transfer time is overlapped with processing time. This is not the case when processing time is faster than transfer time. Overlapping the computation time with the data transfer time is the main challenge for RNN, since RNN computation has little data reuse.

The DeepRnn architecture was implemented on the Zedboard Zynq-7000 SOC XC7Z020 (same from DeepStream). Since DeepRnn is a scalable architecture, it could have different number of MAC units independent of the LSTM model size. A grid of 2×2 MAC units were implemented for timing comparison with the DeepStream, which also runs 4 MAC units in parallel. WM is 16 KB and VM is 2 KB. The total on-chip power is 1.8 W.

V. Experiments

To test the implemented hardware, we trained a character level language model [20] using the training script by Andrej Karpathy written in Torch7. The code can be downloaded from Github[1].

The character level language model predicts the next character given a previous character. Character by character, the model generates a text that looks like the training data set, which can be a book or a large internet corpora with more than 2 MB of words. For this experiment, the model was trained on a subset of Shakespeare's work. The model is composed of 2 LSTM layers, each with 128 hidden units. The batch size was 50, the training sequence was 50 and learning rate was 0.002.

For profiling time, the Torch7 code was ran on other embedded platforms to compare the execution time between them. The Tegra X1 development board and the Odroid XU4 was used in this experiments. Odroid XU4 has the Exynos5422 with four high performance Cortex-A15 cores and four low power Cortex-A7 cores (ARM big.LITTLE technology). The low power Cortex-A7 cores was clocked at 1400 MHz and the high performance Cortex-A15 cores was running at 2000 MHz. A software LSTM implementation was ran on Zedboard's host processor dual ARM Cortex-A9 processor clocked at 667 MHz. Finally, the hardware was ran on the co-processors clocked at 142 MHz.

VI. Results

The use of fixed point Q8.8 data format introduces insignificant quantization error, and comparing the results from the software implementation with the hardware co-processor's output, the average percentage error for the $\mathbf{c}_t$ was 3.9% and for $\mathbf{h}_t$ was 2.8%. Those values are average error of all time steps. The best was 1.3% and the worse was 7.1%. The recurrent nature of LSTM did not accumulate the errors and on average it stabilized at a low percentage. The character level language model generated 1000 characters text, that looks like from Shakespeare's works.

Both the Zedboard Zynq ZC7020 and ZC706 platforms has 4 Advanced eXtensible Interface (AXI) DMA ports available. Each is ran at 142 MHz and send packages of 32 bits. This allows aggregate bandwidth up to 3.8 GB/s full-duplex transfer between FPGA and external DDR3. Figure 3 shows the off-chip memory bandwidth for each design.

The hardware to support the necessary AXIS ports and the CPU interface was measured as the base utilization, which was subtracted with the total utilization of each architecture. Figure 2 shows the utilization from each co-processor.

Figure 4 shows the performance per Watt for different systems running the character generation application, which is a LSTM of 2 layers of size 128. Even with same number of MACs running in parallel and almost same on-chip power consumption, DeepRnn (2×2) achieved lower performance than DeepStream. But the main advantage of DeepRnn is that we can scale the number of MAC units in DeepRnn, while adding more MACs in DeepStream would hit the off-chip memory bandwidth boundary. A projection of DeepRnn (4×4) performance would be close to DeepStore design, which is limited to available internal memory. For larger workloads, Tegra X1 development board achieves performance of 446 Mops/W on a LSTM of 2 layers of size 1024, which is comparable with the DeepRnn design.

[1] https://github.com/karpathy/char-rnn

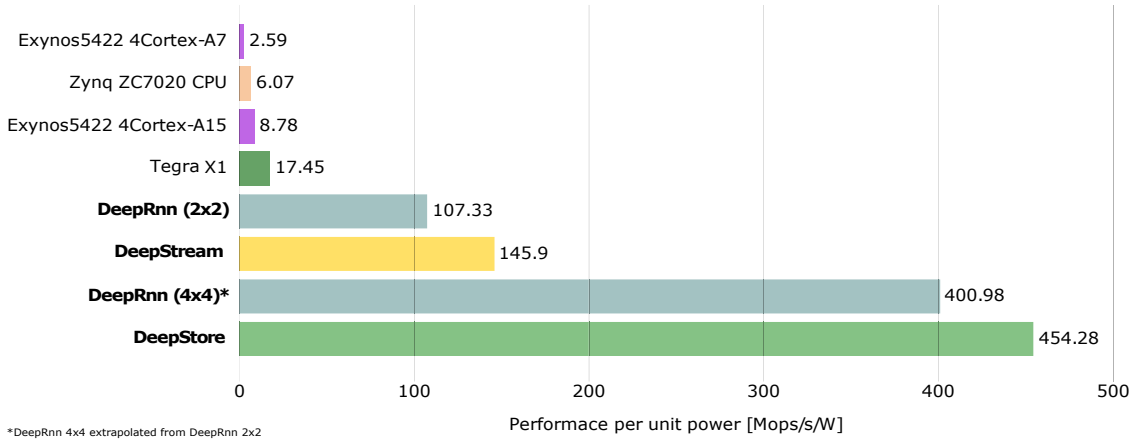

Fig. 4. Performance per unit power running LSTM for different embedded platforms (the higher the better).

VII. Conclusion

This work presented three different hardware implementation strategies of LSTM in FPGA. The hardware successfully produced Shakespeare-like text using a character level model. The study of the various methods to accelerate RNN in hardware provided more insight of the main challenges. The RNNs computation parallelism is hard to be exploit, because of its limited data reuse. Overlapping the computation time with the data transfer time is the main challenge for RNN hardware implementations. Hence, a balance between memory bandwidth and internal store must be achieved for optimal design. Scalability and performance per Watt are also major figures of merit in RNN hardware design. Furthermore, the implemented hardware showed to be significantly faster than other mobile platforms. This work can potentially evolve to a RNN co-processor for future devices, although further work needs to be done.

Acknowledgment

This work was partly supported by Office of Naval Research (ONR) grants N000141210167, N000141512791 and MURI N000141010278 and National Council for the Improvement of Higher Education (CAPES) through Brazil scientific Mobility Program (BSMP). We would like to thank Vinayak Gokhale, Aliasger Zaidy, Alfredo Canziani, Aysegul Dundar, and Jonghoon Jin for the support. We gratefully appreciate the support of NVIDIA Corporation with the donation of GPUs used for this research.

References

[1] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[2] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *IEEE Int. Conf. on ICASSP*, 2013, pp. 6645–6649.

[3] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[4] W. Byeon, M. Liwicki, and T. M. Breuel, "Scene analysis by mid-level attribute learning using 2d lstm networks and an application to web-image tagging," *Pattern Recognition Letters*, vol. 63, pp. 23–29, 2015.

[5] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: A neural image caption generator," *CVPR*, 2015.

[6] H. Fang, S. Gupta, F. Iandola, R. Srivastava, L. Deng, P. Dollár, J. Gao, X. He, M. Mitchell, J. Platt *et al.*, "From captions to visual concepts and back," *arXiv preprint arXiv:1411.4952*, 2014.

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

[8] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[9] *Zynq-7000 All Programmable SoC Overview*, Xilinx, Jan. 2016, v1.9. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[11] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5, pp. 602–610, 2005.

[12] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *ACM Sigplan Notices*, vol. 49. ACM, 2014, pp. 269–284. [Online]. Available: http://dl.acm.org/citation.cfm?id=2541967

[13] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing fpga-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM, 2015, pp. 161–170. [Online]. Available: http://dl.acm.org/citation.cfm?id=2689060

[14] V. Gokhale, J. Jin, A. Dundar, B. Martini, and E. Culurciello, "A 240 g-ops/s mobile coprocessor for deep neural networks," in *IEEE Conf. on CVPRW, 2014*. IEEE, 2014, pp. 696–701.

[15] Y. Guan, Z. Yuan, G. Sun, and J. Cong, "Fpga-based accelerator for long short-term memory recurrent neural networks."

[16] S. Li, C. Wu, H. Li, B. Li, Y. Wang, and Q. Qiu, "Fpga acceleration of recurrent neural network based language model," in *IEEE 23rd Inter. Symp.on FCCM*, May 2015, pp. 111–118.

[17] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," *CoRR*, vol. abs/1602.01528, 2016. [Online]. Available: http://arxiv.org/abs/1602.01528

[18] S. Han, J. Kang, H. Mao, Y. Hu, X. Li, Y. Li, D. Xie, H. Luo, S. Yao, Y. Wang *et al.*, "Ese: Efficient speech recognition engine with compressed lstm on fpga," *arXiv preprint arXiv:1612.00694*, 2016.

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

[20] A. Karpathy, "The unreasonable effectiveness of recurrent neural networks," 2015.