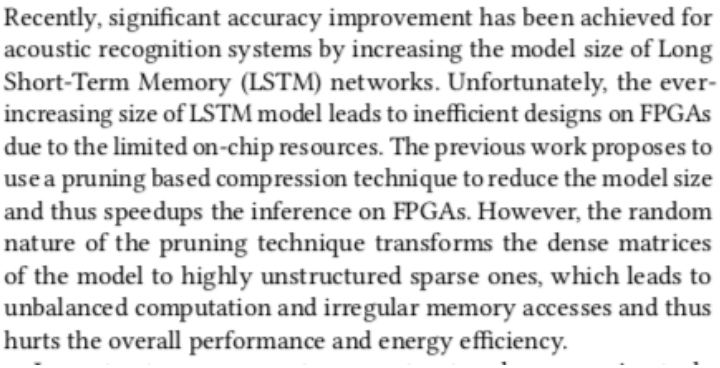
SIAM-IMA Etymo workshop - text extraction

Steven Elsworth

June 13, 2018

Text extraction


Recently, significant accuracy improvement has been achieved for acoustic recognition systems by increasing the model size of Long Short-Term Memory (LSTM) networks. Unfortunately, the ever-increasing size of LSTM model leads to inefficient designs on FPGAs due to the limited on-chip resources. The previous work proposes to use a pruning based compression technique to reduce the model size and thus speedups the inference on FPGAs. However, the random nature of the pruning technique transforms the dense matrices of the model to highly unstructured sparse ones, which leads to unbalanced computation and irregular memory accesses and thus hurts the overall performance and energy efficiency.

Recently, significant accuracy improvement has been achieved for acoustic recognition systems by increasing the model size of Long Short-Term Memory (LSTM) networks. Unfortunately, the ever-increasing size of LSTM model leads to inefficient designs on FPGAs due to the limited on-chip resources. The previous work proposes to use a pruning based compression technique to reduce the model size and thus speedups the inference on FPGAs. However, the random nature of the pruning technique transforms the dense matrices of the model to highly unstructured sparse ones, which leads to unbalanced computation and irregular memory accesses and thus hurts the overall performance and energy efficiency.

Methods

- pdftotxt
- Textract (Backend pdftotxt)
- PyPDF2
- pdfminer
- pyocr (backend tesseract)

See Jupyter Notebook.

Problems faced

Give examples of :

- Equation conversion
- Figure conversion
- White space
- Merged words
- Page columns
- Time

dependencies between operator are complicated. So, it is difficult to evenly allocate computing resources under the FPGA resource constraints while guaranteeing the complex data dependencies.

In this work, we propose to compress the weight matrices in the LSTM inference model in a structured manner by using block-circulant matrix [22]. The circulant matrix is a square matrix, of which each row (column) vector is the circulant reformat of the row (column) vector. Any matrix could be transformed into a set of circulant submatrices a.k.a. block-circulant matrices. Therefore, by representing each block-circulant matrix with a vector, the storage requirement could be reduced from $\mathcal{O}(k^2)$ to $\mathcal{O}(k)$ if the block (vector) size is k. Since the compressed weight matrices are still dense, the block-circulant matrix based compression is amenable to hardware acceleration on FPGAs. In order to further speed up the computation of LSTMs, we propose to accelerate the most computation-intensive circulant convolution operator by applying Fast Fourier Transform (FFT) algorithm to reduce the computational complexity from $\mathcal{O}(k^2)$ to $\mathcal{O}(k\log k)$.

After the model is compressed, we propose an automatic optimization and synthesis framework called C-LSTM to port efficient LSTM designs onto FPGAs. The framework is composed of model training and implementation flows. The former one is in charge of iteratively training the compressed LSTM model and exploring the trade-offs between compression ratio and prediction accuracy. As for the model implementation, it mainly consists of two parts which are (1) template generation and (2) automatic LSTM synthesis framework. For the former part, after analyzing a wide range of LSTM algorithms, we generalize a suite of LSTM primitive operators which is general enough to accommodate even the most complicated LSTM variant [25]. Then, a suite of highly optimized C/C++ templates of the primitive operators are manually generated by walking through a series of optimizations such as datapath and activation quantization, DFT-IDFT decoupling and etc. As for the latter part, the well-trained LSTM inference model is first analyzed and transformed into a directed acyclic dependency graph, where each node represents an operator and each edge indicates the associated data dependency between two operators. Secondly, we propose a specialized pipeline optimization algorithm considering both coarse-grained and fine-grained pipelining schemes to schedule the operators into appropriate stages. In the third step, we use an accurate performance and resource model to enable a fast design space exploration for optimal design parameters. Lastly, the scheduling results and optimization parameters are fed to code generator and backend toolchain as to implement the optimized LSTM accelerator design on FPGAs.

Overall, the contributions of this paper are listed as:

- We employ the block-circulant matrices based structured compression technique for LSTMs which largely reduces the computation complexity and memory footprint without incurring any computation and memory access irregularities. This method results in both compression and acceleration of the LSTM models.
- We develop a general LSTM optimization and synthesis framework C-LSTM to enable automatic and efficient implementations of a wide range of LSTM variants on FPGAs.

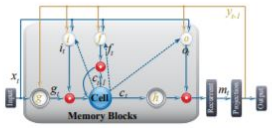


Figure 1: An LSTM based RNN architecture.

The framework mainly consists of a suite of highly optimized C/C++ based templates of primitive operators and an automatic LSTM synthesis flow.

- We present efficient implementations of LSTMs which achieve up to 18.8X and 33.5X gains in performance and energy efficiency, respectively, compared with the state-of-the-art. The proposed implementations incur very small accuracy degradation.

2 LSTM BACKGROUND

LSTM is a key component of the acoustic model in modern large-scale automatic speech recognition (ASR) systems [9, 25], and also the most computation and memory-intensive part. Due to the complicated and flexible data dependencies among gates, cells, and outputs, a lot of LSTM variants have been proposed. In this paper, we use a widely deployed variant called Google LSTM [25] as an example throughout this paper without loss of generality. The architecture details of the Google LSTM is shown in Figure 1. The LSTM accepts an input sequence $\mathbb{X} = (\mathbf{x}_1; \mathbf{x}_2; \mathbf{x}_3; \ldots; \mathbf{x}_T)$ (each of $\mathbf{x}_t$ is a vector corresponding to time t) with the output sequence from last step $\mathbb{Y}^{T-1} = (\mathbf{y}_0; \mathbf{y}_1; \mathbf{y}_2; \ldots; \mathbf{y}_{T-1})$ (each of $\mathbf{y}_t$ is a vector). The input of Google LSTM at time t depends on the output at $t-1$. The LSTM contains a special memory cell storing the temporal state of the network. It also contains three special multiplicative units which are input, output and forget gates. The output sequence $\mathbb{Y} = (\mathbf{y}_1; \mathbf{y}_2; \mathbf{y}_3; \ldots; \mathbf{y}_T)$ is computed by using the following equations iteratively from $t = 1$ to T:

$$\mathbf{i}_t = \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ir}\mathbf{y}_{t-1} + \mathbf{W}_{ic}\mathbf{c}_{t-1} + \mathbf{b}_i), \tag{1a}$$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fr}\mathbf{y}_{t-1} + \mathbf{W}_{fc}\mathbf{c}_{t-1} + \mathbf{b}_f), \tag{1b}$$

$$\mathbf{g}_t = \sigma(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{cr}\mathbf{y}_{t-1} + \mathbf{b}_c), \tag{1c}$$

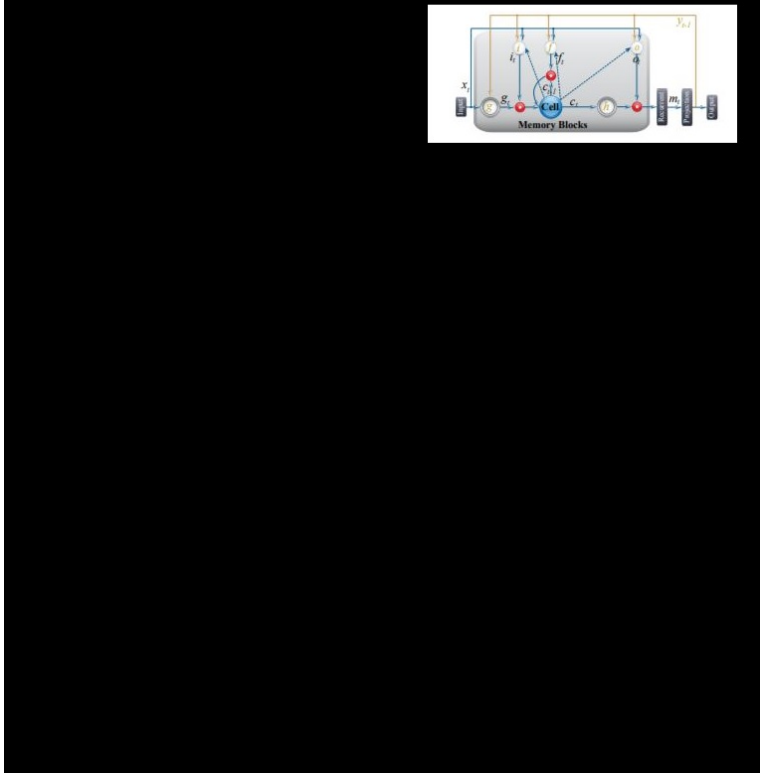
$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{g}_t \odot \mathbf{i}_t, \tag{1d}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{or}\mathbf{y}_{t-1} + \mathbf{W}_{oc}\mathbf{c}_t + \mathbf{b}_o), \tag{1e}$$

$$\mathbf{m}_t = \mathbf{o}_t \odot \mathbf{h}(\mathbf{c}_t), \tag{1f}$$

$$\mathbf{y}_t = \mathbf{W}_{ym}\mathbf{m}_t, \tag{1g}$$

where symbols $\mathbf{i}$, $\mathbf{f}$, $\mathbf{o}$, $\mathbf{c}$, $\mathbf{m}$, and $\mathbf{y}$ are respectively the input gate, forget gate, output gate, cell state, cell output, and a projected output; the $\odot$ operator denotes the element-wise multiplication, and the $+$ operator denotes the element-wise addition. The $\mathbf{W}$ terms denote weight matrices (e.g. $\mathbf{W}_{ix}$ is the matrix of weights from the input vector $\mathbf{x}_t$ to the input gate), and the $\mathbf{b}$ terms denote bias vectors. Please note $\mathbf{W}_{ic}$, $\mathbf{W}_{fc}$, and $\mathbf{W}_{oc}$ are diagonal matrices for peephole connections, thus they are essentially a vector, and the



[1] This estimation considers both weights and indices (there is at least one index per weight after compression in ESE). However, this is a pessimistic estimation for ESE because indices can use fewer bits for representation than weights;

the ADM-7v3 platform. Compared with ESE, we achieve 10.2X and 18.8X performance speedups and 19.1X and 33.5X energy efficiency gains using FFT8 and FFT16, respectively. Since the power consumption of C-LSTM is only half of the ESE, the energy efficiency gain is higher than performance. It is necessary to note that as shown in Table 2, the manufacturing process of XCKU060 FPGA is 20nm while the process of Virtex-7 is 28nm, which means the energy efficiency gain reported here is pessimistic.

Although the promising performance and energy gains are achieved by C-LSTM, the resource utilization for LUT, FF, and BRAM are less than ESE, and more important, the relative PER degradation is very small, which are 0.32% and 1.23% using FFT8 and FFT16, respectively. After detailed analysis, we summarize the fundamental reasons for the high performance and power gains in three aspects. First, the structured compression used in this work eliminates the irregular computation and memory accesses which not only makes the design more regular but also exposes more parallelism. This could be verified in that the DSP resource consumption of the proposed method is much more than ESE. Secondly, the whole model (weights matrices and the projection matrix) could be stored on-chip without fetching data from off-chip DRAM, making the LSTM not bounded by memory. Lastly, the more efficient implementation of LSTM on FPGAs contributes to the high efficiency. For example, we use the 22-segment piece-wise linear function to approximate the activation functions while ESE employs look-up tables which break the activation down into 2048 segments and consume more resources. Moreover, we propose to employ FFT based block-circulant matrix multiplication while ESE uses sparse matrix multiplication which needs to store extra indices for sparse matrices and thus prevents from storing the whole model on-chip.

6.3 Experimental Results of Small LSTM

In order to validate that proposed C-LSTM is not only appropriate for Google LSTM model, we also implement a Small LSTM [20] model on both FPGA platforms.

In KU060 platform, the FFT8 and FFT16 designs could achieve 19.3X and 35.9X performance speedup compared with ESE, respectively. In the ADM-7V3 platform, the performance speedups are 17.5X and 31.9X and the energy efficiency gains are 34.2X and 59.4X compared with ESE, respectively. For both platforms, the PER degradation is 0.29% and 1.16% for FFT8 and FFT16, respectively.

7 RELATED WORK

Recently, FPGA has emerged as a promising hardware acceleration platform for DNNs as it provides high performance, low power and reconfigurability. A lot of FPGA based accelerators have been proposed for convolutional neural networks (CNNs) to overcome the computing and energy efficiency challenges. [28] proposes to utilize systolic array based convolution architecture to achieve better frequency and thus performance for CNNs on FPGAs. [18] employs the Winograd algorithm to reduce the multiplication operators as to save DSP resources and accelerate matrix multiplication in CNNs. [30] proposes to take advantage of the heterogeneous algorithms to maximize the resource utilization for convolutional layers on

This estimation considers both weights and indices (there is at least one index per weight after compression in ESE). However, this is a pessimistic estimation for ESE because indices can use fewer bits for representation than weights; matrix multiplication while ESE uses sparse matrix multiplication which needs to store extra indices for sparse matrices and thus prevents from storing the whole model on-chip.the ADM-7v3 platform. Compared with ESE, we achieve 10.2X and 18.8X performance speedups and 19.1X and 33.5X energy efficiency gains using FFT8 and FFT16, respectively. Since the power