Symbolic Compiler: High Performance LSTM Inference on FPGA

Sixiao Zhu, Ningyi Xu, MSRA

4/15/2016

Overview

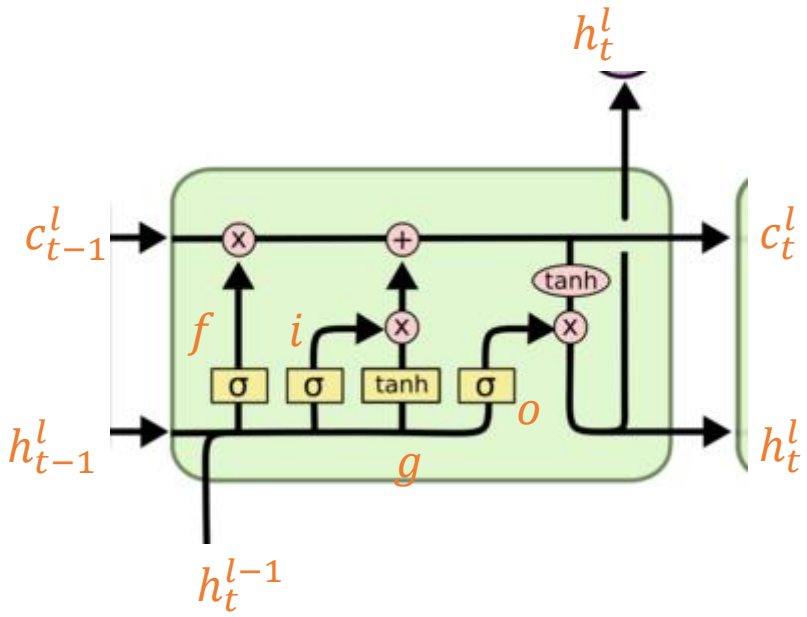
- Why do this?
  - Showing our simboic compiler methodology could support largely heterogeneous neural network structure: CNN, RNN, etc.
  - A single board LSTM prototype for Bing's RNN acceleration plan.
- Model used: Recurrent Neural Network Regularization, Zaremba et al., 2014, a multilayer LSTM, used by TensorFlow as tutorial.

LSTM on pure practical perspective

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W \times \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix}$$

$$c_t^l = f \odot c_{t-1}^l + i \odot g$$

$$h_t^l = o \odot \tanh(c_t^l)$$



Difficulties: IO/Computing matching

- RNN mainly employs $M \times \mathrm{V}$ (hidden state vector multiples weight matrix), this operation is inefficient in terms of data locality, every weight element fetched from DDR is used only once.
- Computing throughput: $1024DSP \times 200MHz = 204.8GB/s$
- DDR bandwidth: $8GB/s$
- **Solution**: batch V, reuse each fetched weight element # batch times. $M \times V \rightarrow M \times \left(V^0, V^1, \ldots, V^{\#batch}\right)$.
- We reuse our GEMM code to implement this actual $M \times M$.

Difficulties: Pipelining the computing process

- In LSTM particularly, $W \times \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix}$ produces gate vectors $\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix}$ as GEMM output.
- This four vectors work together to produce the next c and h.
- Buffering the GEMM outputs requires extra storage and stalls the whole pipeline.
- **Solution**: notice that each gate vector element are used exactly once in element wise operation, we could reorganize our W and h layout to affect the GEMM output order, generating (i_i, f_i, o_i, g_i) together, and consume them immediately.

Crossed vector layout

- This graph illustrates the batching and reorganizing strategies stated in the previous two slides.
- let M be batch size, N be LSTM hidden state dimension.
- We have the format of Matrix A shown right.
- In the right graph, elements are represented this way: $f_{index}^{batch\_id}$.

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} \xrightarrow{in\ GEMM} \begin{bmatrix} i_0^0 & i_0^1 & \dots & i_0^M \\ f_0^0 & f_0^1 & \dots & f_0^M \\ o_0^0 & o_0^1 & \dots & o_0^M \\ g_0^0 & g_0^1 & \dots & g_0^M \\ \vdots & \vdots & \ddots & \vdots \\ i_N^0 & i_N^1 & \dots & i_N^M \\ f_N^0 & f_N^1 & \dots & f_N^M \\ o_N^0 & o_N^1 & \dots & o_N^M \\ g_N^0 & g_N^1 & \dots & g_N^M \end{bmatrix}$$

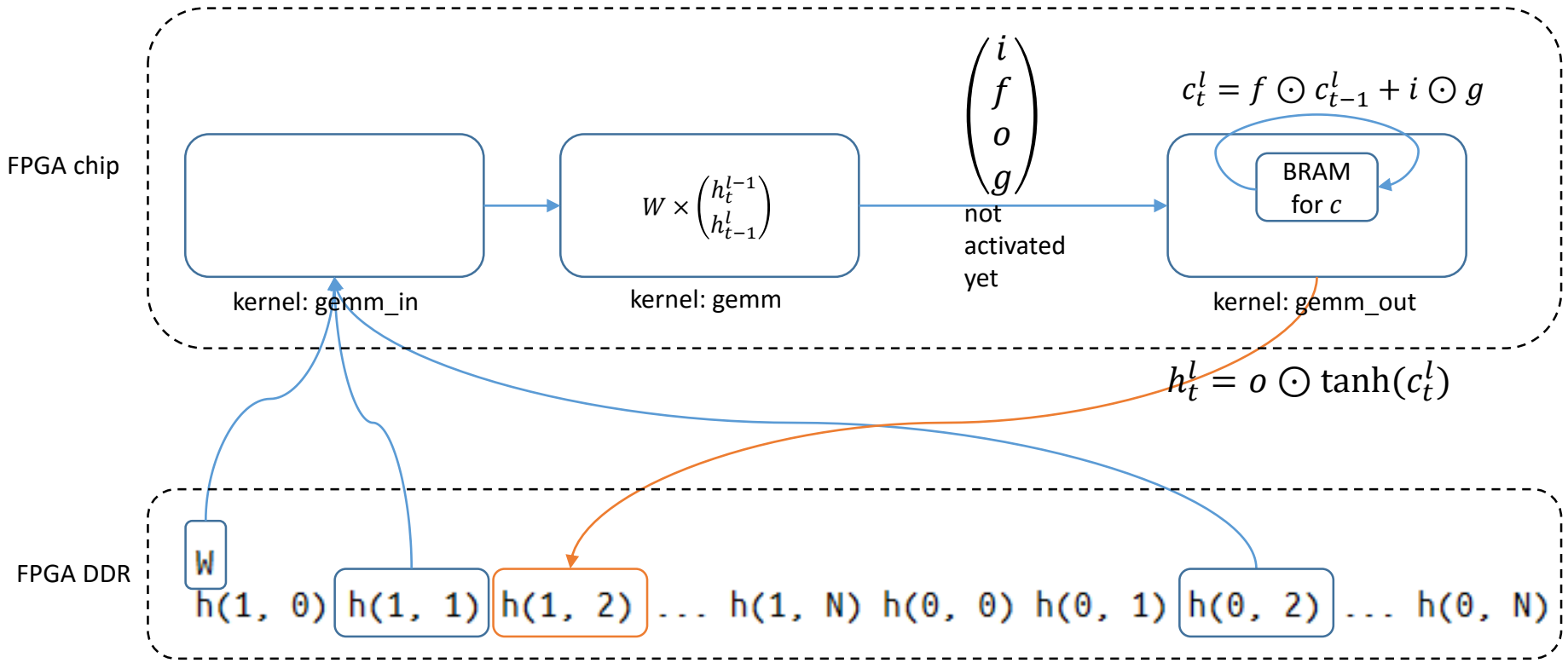
Difficulties: Mapping different computing type to resource as proportion

- The LSTM model used by us uses tanh and sigmoid as activation function, these functions are resource-expensive.
- Activation takes a small proportion of the whole computing.
- Better to use small proportion of resource on activation and large proportion on M*V.
- **Solution**: use a deep fifo between M*V kernel (*gemm*) and the activation kernel (*gemm_out*). Decompose these two process.
- Just need to make sure $\frac{activation\ operation}{activation\ resource} < \frac{M \times V\ operation}{M \times V\ resource}$, in our case, 1536 vector size V.S. 1024 DSP, one copy of activation function is enough.

Extra details

- The activation is done in *gemm_out* kernel in our implementation.
- This kernel is fully pipelined by high level synthesis compiler.
- This means one cycle for three sigmoid and two tanh.
- Notice we implemented float point version of these functions full precision. Consider how much cycles will CPU take compute these. This is the power of FPGA.

Overall diagram



For every h(l, t) in the graph, In fact there are #batch copy of them packed together in memory

Thanks