E-LSTM: Efficient Inference of Sparse LSTM on Embedded Heterogeneous System

Runbin Shi, Junjie Liu, Shuo Wang, Yun Liang and Hayden So
The University of Hong Kong, Peking University
{rbshi, jjliu, hso}@eee.hku.hk, {showang, ericlyun}@pku.edu.cn

Problem
Various models with Long Short-Term Memory (LSTM) network have demonstrated outstanding performance in sequential information processing. Previous LSTM-specific architectures set large on-chip memory for weight storage to alleviate the memory-bound issue and facilitate the LSTM inference in cloud computing. For embedded scenario, however, the resource constraints should be considered while deploying LSTM on CPU-Accelerator heterogeneous system. As follows,

- The weight values should be stored off-chip, even if the compressed sparse model is larger than 1 MB.
- The limited interface bandwidth between CPU and accelerator (64 bit/cycle in E-LSTM) requires a high-efficient data representation of sparse weight matrix and high reuse rate of fetched weight.
- The parallelism obtained from batch-processing is not suitable as it introduces long latency to the system.

Background

LSTM Algorithm

Figure 1: The LSTM-cell accepts the temporal sequence x consisting of embedded vectors (x_t) and outputs sequence h that represents classification/translation/prediction information.

Figure 2: The main computational workload in LSTM cell-iteration is matrix-vector multiplication (W_{xt}, U_{ht}). Strong dependency exists between successive cell iterations.

RISC-V based Heterogeneous System

Figure 3: RISC-V provides ROCC interface for accelerator that facilitates lower latency and smaller chip-area cost.

Implementation

E-LSTM on Github

Implementation methods
- Sparse LSTM benchmark: PyTorch
- LSTM accelerator coupled with RISC-V: cpp model in Spike (RISC-V simulator)

Experiments and Results Comparison

Benchmark Layers

<table>
<thead>
<tr>
<th>Name</th>
<th>Layer</th>
<th>len(x)</th>
<th>len(h)</th>
<th>ts</th>
<th>Sp_{p}</th>
<th>Sp_{s}</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR</td>
<td>LSTMM1</td>
<td>28</td>
<td>28</td>
<td>0.3</td>
<td>0.2</td>
<td>0.02</td>
<td>98.66/98.61/98.11</td>
</tr>
<tr>
<td></td>
<td>LSTMM2</td>
<td>128</td>
<td>128</td>
<td>0.2</td>
<td>0.4</td>
<td>0.29</td>
<td>98.66/98.61/98.11</td>
</tr>
<tr>
<td>LM</td>
<td>LSTMM1</td>
<td>800</td>
<td>800</td>
<td>0.2</td>
<td>0.5</td>
<td>0.56</td>
<td>81.33/81.67/88.52</td>
</tr>
<tr>
<td></td>
<td>LSTMM2</td>
<td>800</td>
<td>800</td>
<td>0.2</td>
<td>0.6</td>
<td>0.41</td>
<td>81.33/81.67/88.52</td>
</tr>
<tr>
<td>LM</td>
<td>LSTMM1</td>
<td>1500</td>
<td>1500</td>
<td>0.4</td>
<td>0.5</td>
<td>0.37</td>
<td>101.63/102.15/106.5</td>
</tr>
<tr>
<td></td>
<td>LSTMM2</td>
<td>1500</td>
<td>1500</td>
<td>0.3</td>
<td>0.4</td>
<td>0.39</td>
<td>101.63/102.15/106.5</td>
</tr>
</tbody>
</table>

Figure 4: In eSELL representation, the non-zero matrix elements of each row in a block are coalesced to a fewer port number of on-chip result buffer. Besides, the encoding of permutation reduces the storage overhead.

Figure 6: In 3-PE case, 2 PEs are stalled because ROCC is used by loading U. Due to the dependency between cell-iterations, U_{ht} can only be computed in sequence. Thus, it becomes the bottleneck in both computation and weight fetching communication.

Cell-fusion based on Inherent h_{t} Sparsity

Figure 5: Both the PE and ROCC interface are fully utilized in 1-PE case.

Figure 7: With the sparse h_{t}, computation workload can be decreased via skipping the U columns corresponding to zeros in h_{t}.

Figure 8: Fine-tuned cell-fusion scheduling effectively utilizes both the PE and ROCC.

Figure 9: Performance with three scheduling schemes.