EcoRNN: Efficient Computing of LSTM RNN on GPUs

Extended Abstract

Bojian Zheng
M.Sc. Candidate, Department of Computer Science
University of Toronto
bojian@cs.toronto.edu

Advisor: Gennady Pekhimenko
Assistant Professor, Department of Computer Science
University of Toronto
pekhimenko@cs.toronto.edu

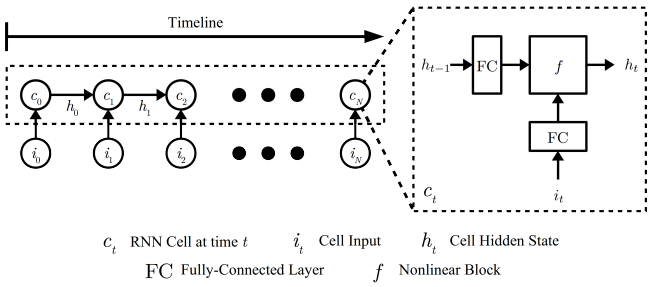

Figure 1: Left: A single-layer LSTM RNN that scans through an input sequence. Right: A zoom-in view of an LSTM cell. Both diagrams have been greatly simplified.

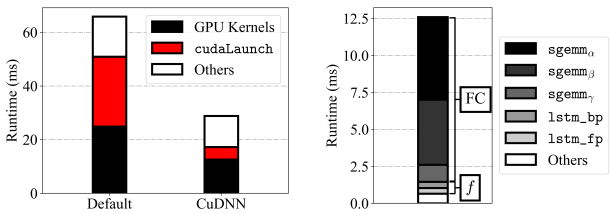

Figure 2: (a) Runtime profile comparison between *Default* and *cuDNN*. *Default* has 5× more cudaLaunch overhead compared with *cuDNN*. (b) Runtime breakdown by GPU kernels in *cuDNN*. All kernel names have been abbreviated.

1 INTRODUCTION

Long-Short-Term-Memory Recurrent Neural Network (LSTM RNN [7], Figure 1) is a state-of-the-art model for analyzing sequential data. Current implementations of LSTM RNN in machine learning (ML) frameworks usually either lack performance or flexibility (i.e. the ability to modify existing computation of LSTM RNN). For example, default implementations (which we will further refer to as *Default*) in Tensorflow [1] and MXNet [3] invoke many tiny GPU kernels, leading to excessive overhead in launching GPU threads (a.k.a. cudaLaunch, Figure 2a). Although *cuDNN* [5], NVIDIA's deep learning library, can accelerate performance by around 2×, it is closed-source and inflexible, hampering further research and performance improvements in frameworks, such as PyTorch [17], that use *cuDNN* as their backend [6]. In this work, we introduce a new RNN implementation called *EcoRNN* that is significantly faster than the open-source implementation in MXNet and is competitive with the closed-source cuDNN. We show that applying data layout optimization, with other techniques that are shown in Appleyard et al.'s work, can give us a maximum performance boost of 3× over MXNet default and 1.5× over cuDNN implementations. We integrate *EcoRNN* into MXNet Python library and open-source it to benefit ML practitioners. We also find that ML compilers [19, 4, 18] can be helpful in making the frontend programming interface more flexible by automatically configuring the data layout to be optimal, which is not possible in cases other than LSTM RNN since it has the property of "optimizing once and for all".

2 KEY OBSERVATIONS

We make the following observations during our optimization of LSTM RNN compututation on the GPUs.

Observation 1. The runtime bottleneck of LSTM RNN is fully-connected (FC) layers. We obtain the runtime breakdown of *cuDNN* GPU kernels using *nvprof* [15], and the result is shown in Figure 2b (due to the fact that *Default* slices the computation of "f" block in Figure 1 into small pieces, its result is difficult to interpret). Figure 2b shows that more than 85% of the time spent on compute has been allocated to matrix multiplies (sgemm is the name for single-precision matrix multiply kernels in cuBLAS library [12]). The annotations beside the stacked bar in Figure 2b group GPU kernels together according to their counterpart in Figure 1, which explains why FC layers in LSTM RNN should be the top candidate for optimization.

Observation 2. Data layout optimization can speed up FC layers. Data layout optimization is a technique that originates from compiler research [9]. The idea behind data layout optimization is that changing data layout (usually from row-major to column-major or vice versa) can result in better locality in the data access pattern, which will further lead to higher cache hit rate, faster memory accesses, and, eventually, better runtime performance. We discover that two computations, $Y = XW^T$ and $Y^T = WX^T$ (Figure 3a), that are mathematically the same can have different cache hit rates. Figure 3b shows an example of such comparison where $X : [64 \times 512]$ and $W : [2048 \times 512]$ (this mimics the FC layers of an LSTM RNN that has batch size and hidden dimension equal to 64 and 512 respectively). We observe that $Y^T = WX^T$ is almost twice as fast as $Y = XW^T$ under this parameter setting, and the reason is that the former has better cache hit rate (50% vs. 10% of latter). Therefore, it can feed data faster into compute units, and thus spend more time in compute rather than waiting for data to arrive from main memory.

3 PRELIMINARY RESULTS

All the experiments are done on a single machine with Intel®Core™i5-3570 [8] CPU and Titan Xp [16] GPU. We have been using CUDA 8.0 [14] toolkit and cuDNN 6.0 [13] for our experiments. To provide fair comparison against *Default* and *cuDNN*, we integrate our implementation of *EcoRNN* into MXNet ver. 0.12.1. We test all three