PERSISTENT RNNS: STASHING WEIGHTS ON-CHIP

Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, Sanjeev Satheesh

ABSTRACT

This paper introduces a framework for mapping Recurrent Neural Network (RNN) architectures efficiently onto parallel processors such as GPUs. Key to our approach is the use of persistent computational kernels that exploit the processor's memory hierarchy to reuse network weights over multiple timesteps. Using our framework, we show how it is possible to achieve substantially higher computational throughput at lower mini-batch sizes than direct implementations of RNNs based on matrix multiplications. Our initial implementation achieves 2.8 TFLOP/s at a mini-batch size of 4 on an NVIDIA TitanX GPU, which is about 45% of theoretical peak throughput, and is 30X faster than a standard RNN implementation based on optimized GEMM kernels at this batch size. Reducing the batch size from 64 to 4 per processor provides a 16x reduction in activation memory footprint, enables strong scaling to 16x more GPUs using data-parallelism, and allows us to efficiently explore end-to-end speech recognition models with up to 108 residual RNN layers.

1 INTRODUCTION

In this paper, we explore techniques for mapping RNNs to throughput optimized processors such as GPUs. We focus on mapping strategies that carefully manage data movement through the processor's memory hierarchy to balance these costs. These changes enable RNN implementations on GPUs that are very efficient at small mini-batch sizes, even on mini-batch sizes of just 4 examples. We exploit this reduction in batch size to decrease the memory footprint of our networks by 16x, allowing us to explore deeper networks without exceeding GPU memory.

To make our results relevant for deployment, we only consider models with a hard constraint of 800ms of future context. We find that accuracy improves with deeper models using batch normalization and skip connections He et al. (2015); Srivastava et al. (2015), reinforcing the trend towards deeper models in vision applications. We present evidence that accuracy continues to improve with increased depth.

2 RNN TO HARDWARE MAPPING STRATEGY

The standard equations for a recurrent network can be written:

$$h_t^l = \sigma(W^l h_t^{l-1} + U^l h_{t-1}^l + b^l) \tag{1}$$

where W^l is the input-hidden weight matrix, U^l is the recurrent weight matrix, h_l^t are the recurrent layer activations and b^l is a bias term.

Implementations of recurrent neural networks typically separate the computation into two stages. The second stage ($U^l h_{t-1}^l$) is computationally more challenging than the first stage ($W^l h_t^{l-1}$) because there is a sequential dependence between timesteps requiring an explicit synchronization between them. Thus, we focus our attention on optimizing this stage by stashing recurrent weights on-chip.

3 IMPLEMENTATION ON A TITANX GPU

The peak floating point throughput of a TitanX running at 1 GHz is 6.144 TFLOP/s. A straightforward implementation of a RNN using GEMM operations achieves 0.099 TFLOP/s at a layer size of

1152 using Nervana Systems GEMM kernels at a mini-batch size of 4. Our initial Persistent RNN implementation with the same layer and mini-batch size achieves over 2.8 TFLOP/s resulting in a 30x speedup.

Our implementation first loads the weight matrix into registers. Then each SM loads all of the input activations from the previous timestep from global memory to shared memory, computes the dot product for each row, performs the nonlinearity, writes the result for the current timestep, and performs a global barrier with all other SMs. The latency required to perform the load operations is approximately four times higher than the time required to perform the math operations for a single timestep. So we break the computation into four independent stages and use software pipelining to overlap the load operation with math, reduce and barrier operations. We use a mini-batch size of four or greater to keep the pipeline full. This computation is implemented with assembly instructions using the MAXAS assembler Gray (2014) to ensure that instructions for each of the four pipeline stages are overlapped.

Synchronization between GPU processors cores is typically achieved implicitly between dependent kernel calls in both CUDA and OpenCL development frameworks. However, this mechanism for synchronization between timesteps requires launching a new kernel that forces the weights to be reloaded from off-chip memory. This causes the synchronization latency of dependent kernels to be approximately 6-10x larger than the time spent performing the math operations for a single timestep, and this cannot be overlapped with computation. We address this problem with an optimized implementation of a global barrier that can be completely overlapped with the math operations for a single timestep.

When training our speech recognition model, we encounter very long utterances that are up to thirty seconds long, corresponding to 3,000 timesteps. For a RNN layer with 1760 hidden units, and a mini-batch size of 64, this corresponds to 1.3 GB of storage per layer. This is much more than the 12.3 MB required to store the layer weights. In practice, with GPUs with 12GB of DRAM, we find that this limits us to networks with about 9 layers. A common solution to this problem is to use truncated back-propagation through time Sutskever (2013) (BPTT). However, we have observed a 20% relative performance degradation of the converged model using this approach, making other techniques that reduce memory footprint, such as reducing the mini-batch size, more attractive.

4 EXPERIMENTS

We evaluate the performance of Persistent RNNs on a large-scale speech recognition task, similar to the Deep Speech 1 Hannun et al. (2014) (DS1) and Deep Speech 2 Amodei et al. (2015) (DS2) systems.

We use the architecture of the DS2 Amodei et al. (2015) as the baseline for these experiments: a recurrent neural network (RNN) trained to ingest speech spectrograms and generate text transcriptions. We evaluate various architectures by varying the number of recurrent layers and the number and span of skip connections between them. We report Word Error Rate (WER) on an English speaker held out development set which is an internal dataset containing 2048 utterances of primarily read speech. We integrate a language model in a beam search decoding step as described in Amodei et al. (2015).

All models are trained for 20 epochs on a 500-hour English dataset. We use synchronous stochastic gradient descent with Nesterov momentum Sutskever et al. (2013) along with a minibatch from the range of $[64, 2048]$ utterances.

Table 1 shows the impact of residual skip connections on very deep RNN architectures with over fifty layers. We find that skip connections are essential for training these models, even when batch normalization is enabled. Models without skip connections fail to converge. For these networks, we find that skipping three or four RNN layers is substantially better than skipping a single layer, and moderately better than any other configuration (except for the outlier of 8, which we cannot explain). This suggests that residual skip connections enable effective optimization of very deep stacks of RNN layers. Preliminary experiments (still running at the time of submission) suggest that the optimal length of skip connections depends on the total depth of the network.

Architecture	Dev (WER)
48 RNN, 61 total, no skip	100.0
48 RNN, 61 total, skip 1	38.77
48 RNN, 61 total, skip 2	33.28
48 RNN, 61 total, skip 3	30.32
8 RNN, 21 total, skip 4	35.69
24 RNN, 37 total, skip 4	31.32
48 RNN, 61 total, skip 4	**29.40**
48 RNN, 61 total, skip 5	29.82
48 RNN, 61 total, skip 6	30.04
48 RNN, 61 total, skip 7	29.87
48 RNN, 61 total, skip 8	**27.44**

Table 1: WER on the internal development set for various strategies of applying skip connections to RNN layers. Skip connections are added between every N RNN layers.

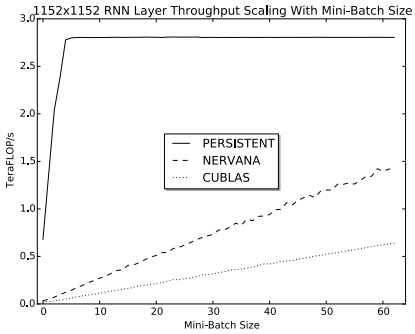

Figure 1: Scaling with mini-batch size.

In this section we compare the computational efficiency of Persistent RNN, against an optimized RNN implementation based on matrix multiplication routines from the NVIDIA and Nervana Systems BLAS libraries. We find that our implementation is substantially more efficient at small mini-batch sizes than either of those implementations. These gains of 30x in the recurrent layers translate into an approximately 10x speedup in the complete system.

Figure 1 compares floating point throughput for Persistent RNN against two other RNN implementations for small mini-batch sizes. Note that after a mini-batch size of four, Persistent RNN consistently deliver approximately 2.8 TFLOP/s, but matrix-multiply based implementations start out much slower, and need relatively large mini batch sizes to become competitive. Even then, we find that layer sizes around 1152 are somewhat too small for matrix multiplication libraries to be efficient, only achieving about 1.5 TFLOP/s at a mini-batch size of 64. Performance is generally much better at layer sizes of 2048 or 2560, suggesting the advantages of persistent RNN implementations will grow as models become relatively deeper and thinner.

Our final experiment evaluates the scalability of Persistent RNNs compared to implementations based on GEMMs on a cluster of GPUs. Using persistent RNNs, we achieve 223 TFLOP/s on 128 GPUs for 24 to 108 RNN layer networks with 1152 hidden units at a mini-batch size of 1024, a 112x speedup over a single GPU. At 128 GPUs, our system sustains about 30% of theoretical peak floating point throughput over an entire training run.

We find both batch normalization and residual skip connections to be effective techniques that allow training deeper RNN models for speech recognition, reinforcing the importance of these techniques that has been previously demonstrated for CNNs applied to vision applications. Our results suggest that training very deep RNNs in excess of 1 PFLOP/s is achievable by future work that focuses on additional kernel level optimizations and scaling to even more GPUs.

5 Conclusion

We demonstrate a technique for achieving high performance for RNN evaluation at very low batch sizes on an NVIDIA TitanX GPU, achieving 2.8 TFLOP/s at a mini-batch size of 4. This provides a 16x reduction in activation memory footprint, and allows us to train models with 108 layers on the same hardware which is 12x deeper than without this technique. We focus our evaluation on unidirectional RNNs with at most 800ms of future context, and demonstrate that accuracy continues to scale with increased depth. This work has shown that some model architectures are constrained not only by hardware performance, but also by the strategy used to map them to hardware. Mapping RNNs to GPUs using matrix multiplication is efficient for shallow networks with large layers, but persistent RNN kernels are much more efficient for deep networks with relatively narrow layers. We expect these gains to directly enable the training of deeper RNN networks on much larger datasets than would be possible without this technique.

REFERENCES

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

Scott Gray. Assembler for nvidia maxwell architecture, 2014. URL `https://github.com/NervanaSystems/maxas`.

Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep speech: Scaling up end-to-end speech recognition. 1412.5567, 2014. http://arxiv.org/abs/1412.5567.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *ArXiv e-prints*, December 2015.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *CoRR*, abs/1505.00387, 2015. URL `http://arxiv.org/abs/1505.00387`.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of momentum and initialization in deep learning. In *30th International Conference on Machine Learning*, 2013.

Ilya Sutskever. Training recurrent neural networks, 2013.