Autogenerating Text via Parallel Implementations of Recurrent Neural Networks

Checkpoint Report

Taylor Clarke (tclarke)
Anirban Ghosh (anirbang)

Our project is to accelerate the training phase of a character based recurrent neural network. The baseline code we are using is based on char-rnn by Andrej Karpathy (https://github.com/karpathy/char-rnn). Our first task was to set up the environment and baseline code in the platforms that we plan to test in. In order to get the baseline code running, we downloaded and installed Torch and Lua libraries on our AFS drives as well as on personal laptops. This step took the most amount of time, as retrieving the various dependencies and linking the various packages was nontrivial.

After setting up the environments, we ran a basic benchmark with all settings set to default on the GHC30 machine's CPU. The RNN took approximately 9 hours to train (detailed testing and settings details below). Once we had the CPU based implementation working, we tried to implement the same RNN on the GHC30's GTX1080 GPU. However, as of now, we are unable to use that GPU since the version of CUDA installed on the GHC30 is an older version that fails to compile the cunn and cutorch libraries for the GTX1080. We are still working on a workaround for this, and plan to test on the latedays cluster in the coming week instead.

To keep in line with our schedule we then implemented the same network on a personal laptop which has a GTX920M with CUDA capability installed. The GTX920M has 2GB of dedicated RAM and 2 multiprocessors with 192 SIMD units per multiprocessor. The training time for the same network and input for the laptop GPU was approximately 33 minutes. We also tried training the network on the same laptop's CPU, which took about 2.5 hours. We think the reason for the laptop CPU outperforming the server is due to the fact that the server is not dedicated for our workload. Given below are the details of our preliminary tests:

- LSTM Parameters
  - RNN Size: 128
  - Number of layers: 2
  - Iterations: 21,150 (for tinyshakespeare)
  - Number of epochs: 50
- GHC30 (CPU): Training time ~9hrs. (tinyshakespeare)
- Local Laptop (CPU: Dual core hyperthreaded Core i7): ~2.5-3 hrs (tinyshakespeare)
- Local Laptop (GPU: NV GTX920M, CUDA 8.0): ~33 mins (tinyshakespeare)
- Data sets: tinyshakespeare

We are now trying to get the same code to run on the Gates and Latedays machines, and hope to get it running on one of these by the end of the week. Our next week's goal also includes porting the code and optimizing it to run on the Xeon Phi. This is one goal that we were supposed to complete last week, but we swapped the GPU implementation and the Phi testing.

We have a couple of concerns, denoted below:

- Driver/CUDA version compatibility with Torch on GHC machines.
- How to get Torch to work with the Latedays cluster - if the installed Torch library will work when the head node of Latedays sends the job to a slave node with the GPU/Phi

Finally, we have also updated our timeline to be more precise:

Nov 21 - Nov 23	Talk with professors and develop final plan for expected deliverables.
Nov 24 - Nov 27	Implement RNN on a Xeon Phi and/or a cluster GPU, depending on talks with professors.
Nov 28 - Nov 30	Tweak parameters of the RNN to test for optimal training time on Phi and GPU.
Dec 1 - Dec 4	Inspect code to investigate potential optimizations specific to Phi and GPU hardware.
Dec 5 - Dec 7	Complete optimizations of hardware and neural network. Begin benchmarking final results.
Dec 8 - Dec 13	Complete benchmarking final results. Create poster and final report. Data on poster and in report will be graph of speedup of parallel implementation as a factor of hardware (Phi or GPU), neural network setup, neural network training specifications, and number of cores.