Optimizing Deep Learning Methods for Image Segmentation with Distributed Training

J. Gregory Pauloski[1], Zhao Zhang[2], Evan Gates[1], Dawid Schellingerhout[1], John D. Hazle[1], David Fuentes[1]

[1]Department of Imaging Physics, University of Texas MD Anderson Cancer Center
[2]Texas Advanced Computing Center

THE UNIVERSITY OF TEXAS MD Anderson Cancer Center
Making Cancer History®

Introduction

Deep learning methods, primarily convolutional neural networks (CNNs), have proven very effective at image classification. CNNs are able to learn complex patterns to identify anatomical structures with high accuracy from medical images. Development of auto-segmentation networks is important to reduce the inherent time requirements and variability of manual segmentation. However, current research is impeded by the computational time required to train and tune the CNNs.

Training a CNN requires significant floating point operations given their approximated $O(n^6)$ time-complexity[1]. While graphics processing units (GPUs) are highly optimized for floating point operations, they have limited RAM capacities and performance scaling beyond a few GPUs. High performance computing environments such as the Stampede2 supercomputer at the Texas Advanced Computing Center allow researchers to execute programs across hundreds of compute nodes with considerably more RAM than a GPU.

Methods

To test the training optimization across multiple compute nodes, we used a 2D CNN U-Net built with the Keras Tensorflow API[2]. This CNN is designed to segment liver lesions from CT scans. The input is 240 x 240 CT scan slices and it outputs images of the same size where each pixel is given a classification corresponding to the segmentation.

The Stampede2 supercomputer has 4,200 Intel Knights Landing nodes each with 68 cores (4 threads per core) and 96 GB of DDR4 RAM plus 16GB of MCDRAM[3]. To take advantage of the available nodes on Stampede2, each node can be provided its own batch to process each step during training. The batch size can also be increased substantially due to the available RAM on each node. This results in a higher effective batch size and decreases the number of steps needed per epoch to process all the training data.

```
initialize workers (nodes)
...
setup model and import data
...
for each epoch {
  broadcast variable states to all workers
  create ImageDataGenerator(batch_size = n)
  while ImageDataGenerator is not empty {
    on each worker:
      get batch from ImageDataGenerator
      train worker on batch
  }
  average gradients across workers
  update model
}
```

Fig. 1: Modified CNN training routine. The ImageDataGenerator Class from Keras controls the process of distributing unique batches to each node. The source code for the CNN with optimizations can be found at https://github.com/gpauloski/livermask.

After processing a batch, each node will update its gradients. To communicate gradients between nodes, we used the Horovod Distributed Learning Framework which implements a wrapper for the CNN optimizer[4]. Horovod averages gradients across nodes with allreduce and then updates the gradients globally. Figure 1 outlines the modified training routine for distributed learning.

Testing was performed with 1, 2, 4, 8, 16, and 32 allocated nodes until the validation loss converged. As a reference, training was also performed on a Nvidia Kepler Titan 6GB and Quadro P5000 12GB. A dataset of 14,909 images was used with 90% of the images allocated for training and the remaining 10% used for validation.

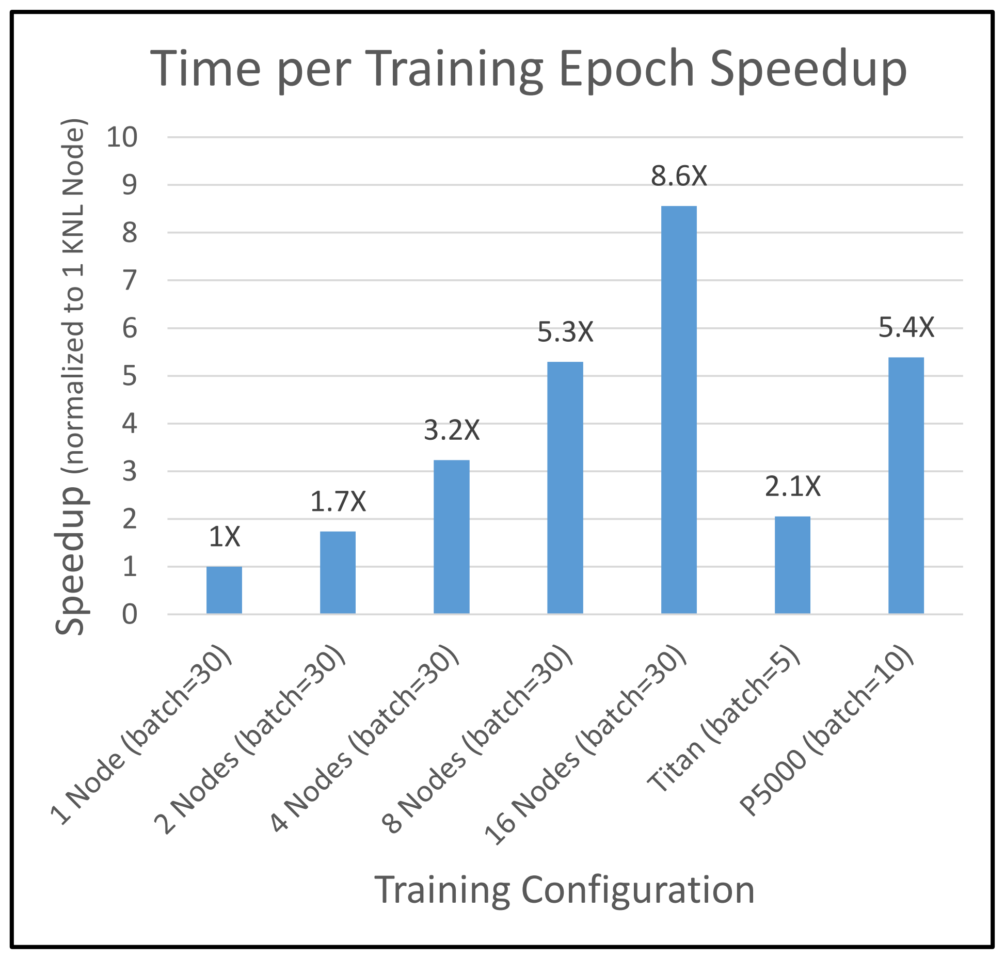

Fig. 2: Speedup of the time to complete one epoch during training normalized to 1 KNL node. For small numbers of nodes (<4), scaling is approximately linear but as the node count increases, scaling performance diminishes. The largest batch size possible given the available RAM was used on the KNL nodes as well as the reference Nvidia GeForce Titan 6GB and Quadro P5000 12GB.

Results

Increasing the effective batch size by providing each node a unique batch to process reduced the average time per epoch to process all of the training data as seen in figure 2. Time per epoch using 8 KNL nodes is comparable to the P5000, and with 16 nodes, the time per epoch improved by 60% over the GPU.

To understand the impact of averaging gradients across the nodes, we plotted the training loss over time in figure 3. Performance gains are again clear moving from 1 node up to 8 nodes. At 16 nodes, performance decreases in the early stages of training. Furthermore, 32 nodes performed substantially worse indicating the performance loss from gradient communication between nodes became greater than the performance gain from increased batch size.

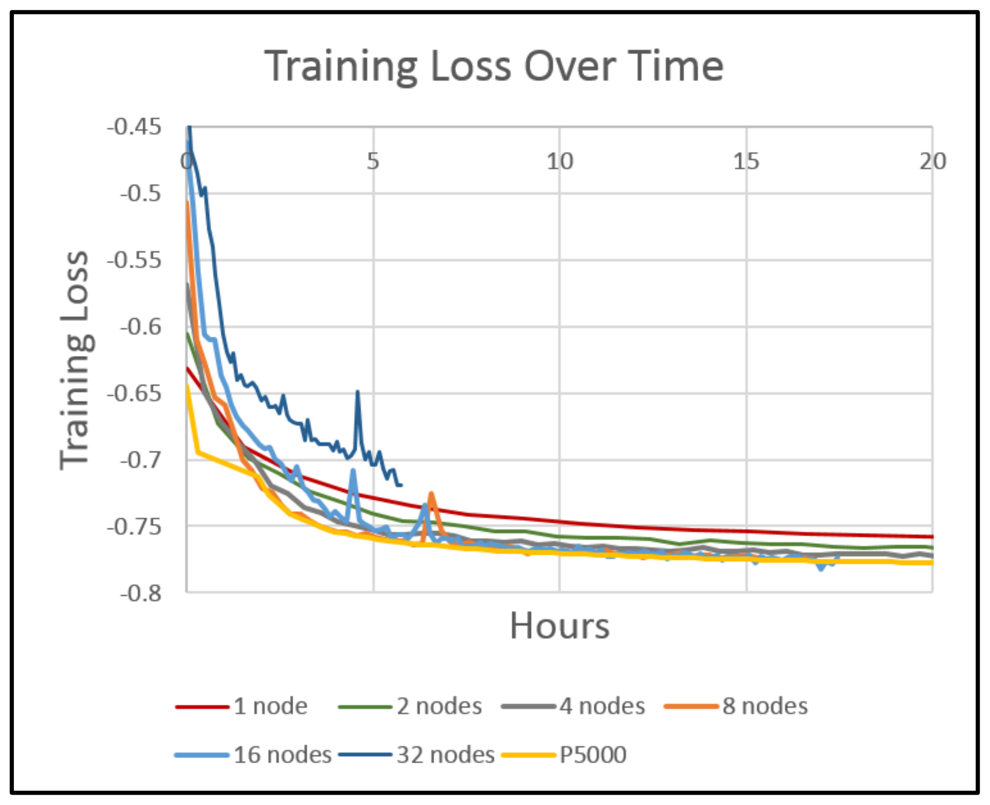

Fig. 3: Training loss over time. Performance improve as node count increases up to 8/16 nodes which perform on par with the P5000. Beyond 16 nodes, performance decreases, and in many cases the loss will not converge.

Figure 4 depicts the Dice score on the reserved data for validation throughout the training session. While figure 2 shows time per epoch scaling well with nodes, the efficiency of each epoch decreases in terms of increasing the validation dice score. Increasing the effective batch size is known to reduce a models ability to generalize and converge[5]. This can often be

Node Count	1	2	4	8	16	P5000
Loss = -0.75 (hours)	12.01	7.45	4.65	3.50	4.91	3.47
Loss = -0.76 (hours)	22.66	12.40	7.60	7.00	6.83	5.42

Table 1: Hours required to reach training losses on -0.75 and -0.76. 16 nodes performs worse than 8 nodes until later in training. 32 nodes was excluded because it failed to converge.

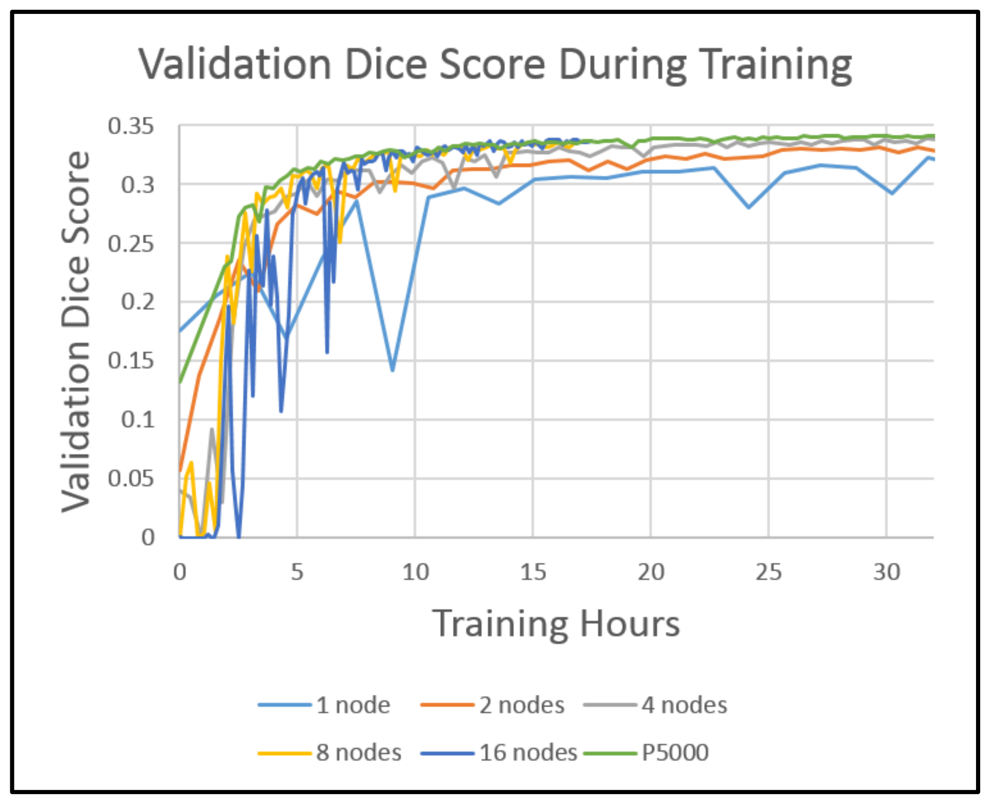

Fig. 4: Increasing variability at higher node counts is due to the validation being done independently on each node before the allreduce step.

compensated for by scaling the learning rate by the node count, however, convergence issues were still present.

Conclusions

Distributed learning using multiple CPU nodes in high performance computing environments scales well at low node counts. As CNNs used in research become larger and more complex, the need for computing environments with large amounts of RAM become more important. The initial results presented are promising for optimizing the training of CNNs in high performance computing environments.

To outperform GPUs, further work needs to be done to understand the balance between the positive and negative impacts of increase batch size. Furthermore, network complexity and parameter optimization such as learning rate reduction schemes could have notable impacts on the performance. Careful consideration should also be given to the complexity of implementing these optimization strategies in existing neural networks to ensure researchers can take advantage of the benefits.

References

1) He, K., & Sun, J. (2015). Convolutional neural networks at constrained time cost. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5353-5360.
2) https://github.com/gpauloski/livermask
3) https://portal.tacc.utexas.edu/user-guides/stampede2
4) Sergeev, A., & Balso, M.D. (2018). Horovod: fast and easy distributed deep learning in TensorFlow. CoRR, abs/1802.05799.
5) Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy & Ping Tak Peter Tang (2016). On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. CoRR, abs/1609.04836.