Improving Strong-Scaling of CNN Training by Exploiting Finer-Grained Parallelism

Authors: Nikoli Dryden, Naoya Maruyama, Tom Benson, Tim Moon, Marc Snir, Brian Van Essen

Presented by: Kinan Alattar
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
  • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Motivation

• There is a need to scale CNN training, mainly to keep up with
  • Growing datasets
  • Reducing training time
• Datasets with very large samples have large memory requirements
• Growing popularity of using deep learning in new domains which incentivizes for a push to continued scaling and performance improvements.
• Take advantage of significant compute power available to researchers and so on ...
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
  • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Existing Approaches & Their Limitations

• Existing frameworks typically use a data-parallel approach to scaling
  • Where the samples in a mini-batch are partitioned between processors
• This approach is limited by the number of samples and in a mini-batch, which can be difficult to increase due to generalizations and convergence issues
• Memory consumption for large samples becomes problematic
Outline

• Motivation
• Existing Approaches & Their Limitations
  • Proposed Approach
    • Sample and spatial parallelism algorithms
    • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Proposed Approach

• Conceptually, a convolutional layer can be specified by five dimensions,
  • The dimensions being: samples, height, width, channels, and filters.
• new approaches to convolution which parallelize using spatial decomposition or a combination of sample and spatial decomposition
• This enables additional parallelism with the cost of extra communication in forward and back propagation.
  • which can make them less advantageous in some situations compared to pure sample parallelism
Proposed Approach (Sample Parallelism)

• At a high level, sample parallelism partitions x, y, dL/dy and dL/dx along the N dimension, assigning complete samples to processors
• The weights are stored redundantly on every processor
• Computation of y (forward propagation), dL/dx, and the local contributions of dL/dw (backpropagation) can then be computed independently
• Allreduce is required to complete the sum to form the final dL/dw, after which SGD can proceed independently on each processor
Proposed Approach (Spatial Parallelism)

• Spatial parallelism is more complicated, the spatial dimensions H and W of $x$, $y$, $\frac{dL}{dy}$, and $\frac{dL}{dx}$ are partitioned along processors.

• Most of forward propagation can be performed locally,
  • but when a filter of size greater than $1 \times 1$ is placed near the border of a partition, remote data will be needed to compute the convolution.

• And so a small number of rows and/or columns will need to be communicated from the remote procs.
  • This is done using a halo exchange.

• Backpropagation is similar,
  • requiring a halo exchange on $\frac{dL}{dy}$ to compute $\frac{dL}{dx}$, and using the data from forward propagation to compute the local contributions of $\frac{dL}{dw}$.

• Finally, like in sample parallelism, an allreduce computes the sum in $\frac{dL}{dw}$. 
Proposed Approach (Continued)

Fig. 1. (a) Forward and backpropagation phases (red and blue arrows) for sample parallelism with two processors and a global mini-batch of size 4. (b) Example halo exchange for spatial parallelism on four processors. The solid red box is where the $3 \times 3$ convolution filter is centered and red arrows indicate data movement. The shaded region is the entire halo region.
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
    • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Implementation - extending LBANN

• The algorithms presented so far are implemented by extending the LBANN toolkit
  • LBANN Provides an underlying substrate for MPI-based parallel training with GPU acceleration

• A small C++ library for distributed tensor data structures was developed to extend LBANN for fine-grained distribution convolutions
  • Provides high-level abstractions for common tensor primitives used in CNN training
A basic set of layers used in typical CNNs, including convolutions, pooling, batch normalization, and ReLU are also implemented.

The paper focuses on cluster systems with NVIDIA GPUs as the main computing platform for training and uses NVIDIA’s cuDNN library for optimized compute kernels.

Since cuDNN is not aware of the distribution of tensors,
  • the library performs halo exchanges before convolutions and pooling.
Implementation - extending LBANN (Optimizing for strong scaling)

• Fine-grained parallelization allows for use a larger set of parallel resources,
  • And, so, careful optimization of data movement becomes more important, especially for strong scaling

• In spatial partitioning, halo exchanges are required and can be prohibitively expensive

• Technique of overlapping halo exchanges with independent computations

• The latencies of halo exchanges can also be minimized by using asynchronous low-latency communication mechanisms such as GPUDirect RDMA (if it's available)
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
  • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Performance model

• The new approach complicates parallelizing a network to achieve good speedup on a system

• Performance model that when given:
  • a platform and a CNN architecture, is able to automatically find good parallel execution strategies, accounting for memory requirements
Performance model (Continued)

• It may be advantageous to use different distributions of data for different layers in a CNN, in order to parallelize them differently.
  • For example, spatial parallelism is unlikely to provide a significant benefit to a layer with a small spatial domain.

• This introduces a number of knobs that can be tuned to parallelize a network

• Selecting, for each layer, the appropriate parallelization scheme is difficult.

• Present a simple optimization approach for selecting good parallel execution strategies using the performance model
Performance model (Parallel execution strategies)

• First candidate distributions are generated for each layer.
• For convolutional layers,
  • heuristically select distributions that are load balanced and prefer cheaper partitioning methods (i.e. sample over spatial parallelism) when possible.
• Given the set of candidate distributions for each layer, find the best parallel execution strategy for the CNN by reducing to the single-source shortest path problem
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
  • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Evaluations

• The algorithms are evaluated via microbenchmarks and end-to-end training.

• a fully-convolutional ResNet50 for ImageNet-1K classification

• The data consists of images representing a hydrodynamics simulation state at a timestep, and the problem is to predict, for each pixel, whether the mesh cell at that location needs to be relaxed to prevent tangling
Evaluations (Continued)

For $N = 1$ for conv1 forward propagation does not scale well due to limited computation to hide halo exchanges.

Backpropagation fares better, and results in net improvements to forward and backpropagation of $\sim 1.35x$ with 8 GPU.

Res3b_branch2a is a $1 \times 1$ convolution from the middle of ResNet, with a fairly small spatial domain.

The filter size means that no halo exchange is needed, avoiding communication overheads.

Forward propagation does not show significant performance improvements beyond two GPUs, due to fixed kernel overheads.

Performance degrades somewhat with 16 GPUs, due to

Backpropagation shows improvements up to 16 GPUs except that the 2 GPUs/sample case is significantly slower than 4 GPUs/sample at 4 GPUs due to the performance of the underlying cuDNN kernels.

Fig. 2. Microbenchmark results for layers conv1 and res3b_branch2a of ResNet-50 comparing parallelization schemes in forward (FP) and backpropagation (BP). Error bars are ± one standard deviation. Black shapes are performance model predictions. Specifications for each layer are above each figure.
Evaluations (Continued)

Using spatial parallelism we achieve 1.4x speedups with 2x as many GPUs, and up to 1.8x with 4x as many GPUs.

Comparing pure sample parallelism to hybrid sample 2-way and 4-way parallelism, speedups decrease slightly at larger scale for more extensive decomposition, due to the implementation being unable to fully overlap the cost of allreduces.

### TABLE III

<table>
<thead>
<tr>
<th>N</th>
<th>Sample (32 samples/GPU)</th>
<th>Hybrid (32 samples/2 GPUs)</th>
<th>Hybrid (32 samples/4 GPUs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0.106s</td>
<td>0.0734s (1.4x)</td>
<td>0.0593s (1.8x)</td>
</tr>
<tr>
<td>256</td>
<td>0.106s</td>
<td>0.0732s (1.4x)</td>
<td>0.0671s (1.6x)</td>
</tr>
<tr>
<td>512</td>
<td>0.105s</td>
<td>0.0776s (1.4x)</td>
<td>0.0617s (1.7x)</td>
</tr>
<tr>
<td>1024</td>
<td>0.105s</td>
<td>0.0747s (1.4x)</td>
<td>0.0672s (1.6x)</td>
</tr>
<tr>
<td>2048</td>
<td>0.108s</td>
<td>0.0733s (1.5x)</td>
<td>0.0651s (1.7x)</td>
</tr>
<tr>
<td>4096</td>
<td>0.0984s</td>
<td>0.078s (1.3x)</td>
<td>0.066s (1.5x)</td>
</tr>
<tr>
<td>8192</td>
<td>0.109s</td>
<td>0.0785s (1.4x)</td>
<td>0.0725s (1.5x)</td>
</tr>
<tr>
<td>16384</td>
<td>0.108s</td>
<td>0.0844s (1.3x)</td>
<td>0.0792s (1.4x)</td>
</tr>
<tr>
<td>32768</td>
<td>0.109s</td>
<td>0.0869s (1.3x)</td>
<td>n/a</td>
</tr>
</tbody>
</table>
Outline

• Motivation
• Existing Approaches & Their Limitations
• Proposed Approach
  • Sample and spatial parallelism algorithms
  • Implementation (extending LBANN)
• Performance model
• Evaluations
• Summary
Summary

• Algorithms for parallelizing convolutional layers by exploiting parallelism available from sample and spatial decompositions, and describes extensions to channel and filter decompositions.

• Performance model that reasonably approximates observed performance and provides intuition about different approaches to parallelizing convolution.

• Comprehensive evaluations to these implementations with microbenchmarks and end-to-end training.

• Approaches for extracting parallelism from convolutional layers for training CNNs, enabling improved strong scaling.

• The feasibility of training on a dataset with very large samples
  • which would otherwise be infeasible due to memory constraints
That's all! -- Any questions?