MYTH BUSTED: GENERAL PURPOSE CPUS CAN'T TACKLE DEEP NEURAL NETWORK TRAINING – PART 2

Written by Pradeep Dubey | November 12, 2015

Multi-Node Caffe* Training on Intel Xeon Processor E5 Series

In the second installment of the Intel® Math Kernel Library technical preview package, we present an optimized multi-node implementation using Caffe* that builds on our previous release of an optimized single node implementation. This implementation scales upto 64 nodes of Intel® Xeon® processor E5 (Intel® microarchitecture code name Haswell) on the AlexNet neural network topology, and can train it to 80 percent Top-5 accuracy in roughly 5 hours, using synchronous minibatch stochastic gradient descent. Below is a view into the technical details of how we achieved such amazing strong scaling for this very difficult problem.

MULTI-NODE SYNCHRONOUS SGD

In this work we perform strong scaling of the synchronous mini-batch stochastic gradient descent algorithm. We scale the computation of each iteration across multiple nodes, such that the multi-threaded, and multi-node parallel implementation is equivalent to a single-node single-threaded serial implementation. We utilize data- and model-parallelism, and a hybrid parallelism approach to scale computation. We present a detailed theoretical analysis of computation and communication balance equations, and determine strategies for work partitioning between nodes.

BALANCE EQUATIONS IN DATA PARALLELISM

Clearly, the amount of computation in the number of floating-point operations (FLOPS) in this layer for a forward pass is:

Computation_FPROP = 2 × ifm × ofm × kernel_w × kernel_h × output_w × output_h

Recall that the computation for forward propagation, backward propagation and weight gradient calculation is the same. Now if we consider a multinode implementation where the number of data-points assigned per node is MB_node, then the total computation per node, per iteration is: Computation = 2 × ifm × ofm × kernel_w × kernel_h × output_w × output_h × 3 × MB_node The total communication per iteration can similarly be estimated for a **data-parallel approach**. In each iteration, the partial weight gradients must be communicated out of the node, and the update weights should be received by each node. Hence the total communication volume is:

Communication = data_size × ifm × ofm × kernel_w × kernel_h × (1 + (1 - overlap))

Here overlap is the amount of overlap afforded by the software/algorithm between the sends and receives. Assuming floating point data representation, and complete overlap (overlap = 1) of sends and receives, we can estimate the communication volume (in bytes) to be:

Communication = 4 × ifm × ofm × kernel_w × kernel_h

The communication-to-computation ratio for data parallel implementation of a single layer is therefore computed as:

Algo-comp-to-comm-ratio = 1.5 × output_w × output_h × MB_node

It is notable that the algorithmic computation-to-communication ratio does not depend on the kernel size or number of input and output feature maps or stride, but instead solely depends on the size of the output feature-map and the number of data-points assigned per node.

For the neural network training computation to scale, the time taken for computation should dominate the time for communication. Hence the algorithmic computation-to-communication

Let us consider the implications of this observation for three cases and three hardware options, one for an Intel Xeon processor with an FDR InfiniBand* link, another for an Intel Xeon processor with 10GigE Ethernet, and another for a dense compute solution like Intel® Xeon PhiTM processor with Intel® Omni-Path Fabric. First let us consider the three layers we want to study:

1. A convolutional layer with 55×55 output feature map (like C1 layer of AlexNet, or similar to C2 layer of VGG networks) with algorithmic-compute-to-communication ratio of: 4537×MB_nod

2. A convolutional layer with 12×12 output feature maps like C5 in OverFeat-FAST (and which constitutes the bulk of OverFeat-FAST computation), where the algorithmic computation-to-communication ratio is: 216×MB_node

3. A fully connected layer which can be considered as a convolutional layer with feature map size = 1, where the algorithmic compute-to-communication ratio is 1.5×MB_node

It is notable that the aforesaid algorithmic compute-to-communication ratios are optimistic and best-case scenarios. The worst case scenario happens when overlap=0, and then these values are halved. For example, the ratio for fully connected layers becomes 0.75×MB_node. It is notable that these are theoretical analysis, and both the computation as well as communication times may vary in an actual implementation.

Now let us consider the system computation-to-communication ratios for the three hypothetical platforms described earlier:

1. A server class CPU C1 (with 2.7TF peak SP performance), with FDR InfiniBand = 2700GFLOPs/7GB/s = 386.
2. Same server class CPU C1, with Ethernet = 2700/1.2GB/s = 2250
3. A manycore processor M1 (with 6.0TF peak SP performance) with Omni-Path Fabric/PCI Express* Gen 3 = 6000GFLOPs/12.5GB/s = 480

can estimate the minimum number of data points which can be assigned to each node. This in conjunction with the size of the minibatch, sets limits on the scaling possible for data-parallel approach to neural network training.

	Intel® Xeon® processor + InfiniBand FDR	Intel® Xeon® processor + 10Gb Ethernet	Intel® Xeon Phi^{TM} + Omni-Path Fabric
C1 (55x55)	1	1	1
C5 (12x12)	2	11	3
F1 (1x1)	258	1500	320

Figure 1. The minimum number of data points which must be assigned to a given node.

Clearly there are several cases where an inordinately large number of data points must be assigned to a given node in order to make data-parallelism beneficial. Often this is greater than the size of the mini-batch needed to converge at a reasonable rate. Hence, the alternative method of model-parallelism is needed to parallelize neural network training.

MODEL PARALLEL APPROACH

Model parallelism refers to partitioning the model or weights into nodes, such that parts of weights are owned by a given node and each node processes all the data points in a mini-batch. This requires communication of the activations and gradients of activations, unlike communication of weights and weight gradients as is in the case of data parallelism.

For analyzing model parallelism, we should note that the forward and back-propagation need to be treated differently. This is because during the forward propagation we cannot overlap communication of the previous layer activations with the forward propagation operation of the

ANALYZING THE MODEL PARALLEL APPROACH:

We first consider a simple model parallel approach where each node operates on a part of the model of size: ifm_b×ofm_b input- and output-feature maps. In this case, the computation for the forward pass, or backward-pass, or weight-gradient update is given as:

Computation = 2 × ifm_b × ofm_b × kernel_w × kernel_h × output_w × output_h × minibatch

For the forward pass the amount of data received by this layer is:

Recv_comms = 4 × ifm_b × input_w × input_h × minibatch × (ifm/ifm_b - 1)

The amount of data sent out by the previous layer is:

Send_comms = 4 × ifm_b× input_w × input_h × minibatch

Hence the time taken for a forward pass with no compute and communication overlap for a given layer is:

Computation/System-flops + (Recv_comms + Send_comms)/Communication-bandwidth

Similar to the analysis of data-parallel multinode implementations, we can compare the communication and computation in the model parallelism. The algorithmic compute-to-communication ratio is:

2 × ifm_b × ofm_b × kernel_w × kernel_h × output_w × output_h × minibatch/ 4 × ifm × input_w × input_h × minibatch

This can be simplified as: 0.5 × ifm_b × ofm_b× kernel_w × kernel_h × feature-size-ratio/ifm (here feature size ratio is the ratio of size of output feature to input feature). This ratio is independent of the mini-batch size. The algorithmic ratio can be further simplified to: 0.5 × ofm× kernel_w × kernel_h × feature-size-ratio/NUM_NODES (NUM_NODES = (ifm*ofm)/(ofm_b*ifm_b)). We then consider mirrored operations for backpropagation and no

0.75 × ofm× kernel_w × kernel_h × feature-size-ratio/NUM_NODES > system-compute-to-comm-ratio

Exploring this limit for C5 layer described earlier, and Intel microarchitecture code name Haswell processors with FDR-IB we obtain the following:

0.75 × 1024 × 9 × 0.73 /NUM_NODES > 386, so NUM_NODES < 14.

Similarly for a fully connected layer with 4096 output feature maps we have the following conclusions: 3072/NUM_NODES > 386, so NUM_NODES < 8

Clearly model parallelism alone does not scale well to multiple nodes even for convolutional layers. However, the choice of parallelization strategy is also dictated by which of model and data parallelism works better for a given layer. In particular, if we compare data and model parallelism for a 4096-4096 fully connected layer, we can easily draw a conclusion that model parallelism scales several times better than data parallelism. In particular, for a mini-batch size of 256, a fully connected layer cannot even scale beyond one node using data-parallelism. However, we must highlight the challenges in software design needed to overlap computation and communication in model-parallelism.

There is therefore a clear need to have both data parallelism and model parallelism for different types of layers. Of particular interest therefore is the question: “When to use model parallelism and when to use data parallelism?” This is answered by simply comparing the volume of data communicated in both schemes. The ratio of communication volume in model and data parallelism is:

(1.5 × output_w × output_h × MINIBATCH/NUM_NODES)/(0.5 × ofm× kernel_w × kernel_h × feature-size-ratio/NUM_NODES)

We can simplify this ratio to be dependent on the MINIBATCH size and surprisingly independent of the number of nodes the problem is mapped to. One should pick model parallelism over data parallelism if:

Consider now the fully connected layer F1, where ofm=3072 and input_w/h kernel_w/h are all 1. The equation above indicates that model parallelism is favored as long as MINIBATCH is less than 1024. In visual understanding neural networks, MINIBATCH is less than or equal to 256, hence for fully connected layers we use model parallelism, while for convolutional layers we use data parallelism. In ASR networks MINIBATCH is often larger than 1024, so data parallelism is the preferred route for this case.

In the tech preview package we focus on convolutional neural networks, and perform data parallelism for convolutional layers and model parallelism for fully connected layers. This is aligned with the method proposed by Alex Krizhevsky in his paper [http://arxiv.org/abs/1404.5997]

A special thank you to Dipankar Das, Karthikeyan Vaidyanathan, Sasikanth Avancha and Dheevatsa Mudigere from Intel Lab's Parallel Computing Lab and Vadim Pirogov from Intel's Software and Services Group. They continue to be the driving force behind the research and performance optimizations work illustrated in this blog.

November 12, 2015 [https://itpeernetwork.intel.com/myth-busted-general-purpose-cpus-cant-tackle-deep-neural-network-training-part-2/] Pradeep Dubey Big Data and Analytics Big Data, Cloud, Code Modernization, Intel Xeon E5, Intel Xeon Phi

ABOUT PRADEEP DUBEY

Intel Fellow at Intel Labs

View all posts by Pradeep Dubey ⟶