Parallelizing Strategies for Convolutional Neural Networks

Introduction

Convolutional networks popularly known as ConvNets have revolutionized the approach of solving multiple machine learning problems specially in the widely used fields like computer vision , natural language processing , speech processing and other engineering fields. Unlike neural networks Convolutional networks are capable of learning higher order spectral features from the data. The learning is done through a series of convolutional layers with nonlinear and/or pooling layers in between them with fully connected layers at the end. This series of layers are responsible for converting raw data into spectral features as they pass through the series of layers. However as the layers increases , the time complexity and model complexity increases. This in turn increases the training time as well as the memory size of the model. For reasonably large data sets , commodity computing system might not work due to memory constraints or even if it works, it takes a huge amount of time to train, hence a cluster based training is an ideal choice. Popularity of ConvNets have influenced people to take advantage of accelerator based parallel computing platforms like CUDA, OpenMP etc… . Caffe is one of the framework which takes advantage of GPU Infrastructure provided by NVIDIA.

For training the ConvNets using high performance computing clusters, several programming models like MPI, OpenSHMEM etc… can be used. Since Cuda Aware MPI which makes use of computational capabilities of nodes as well as GPUs has shown good results , the same can be employed to address the training of ConvNets. This work is to train the ConvNets using GPU based clusters and explore some of the optimizations to improve training time without sacrificing the accuracy of the model. In order to reduce the training time by parallelizing the training , two methodologies are explored namely Data parallel and Model parallel using two dimensions of training namely training data and ConvNet model respectively.

Data Parallelism

Data parallelism involves the strategy of training ConvNet by distributing training data using multiple nodes. Since the forward pass in the training just requires the model parameters and batch of data-set, every node is assigned with the same model but with a different data set.
However once the errors are established in the nodes, all the errors are reduced to one common average error and the corresponding updates needs to done to the model in backward pass. In the backward pass only one of the nodes is sufficient . Once the model is updated the same can be repeated as next iteration using a new set of data.

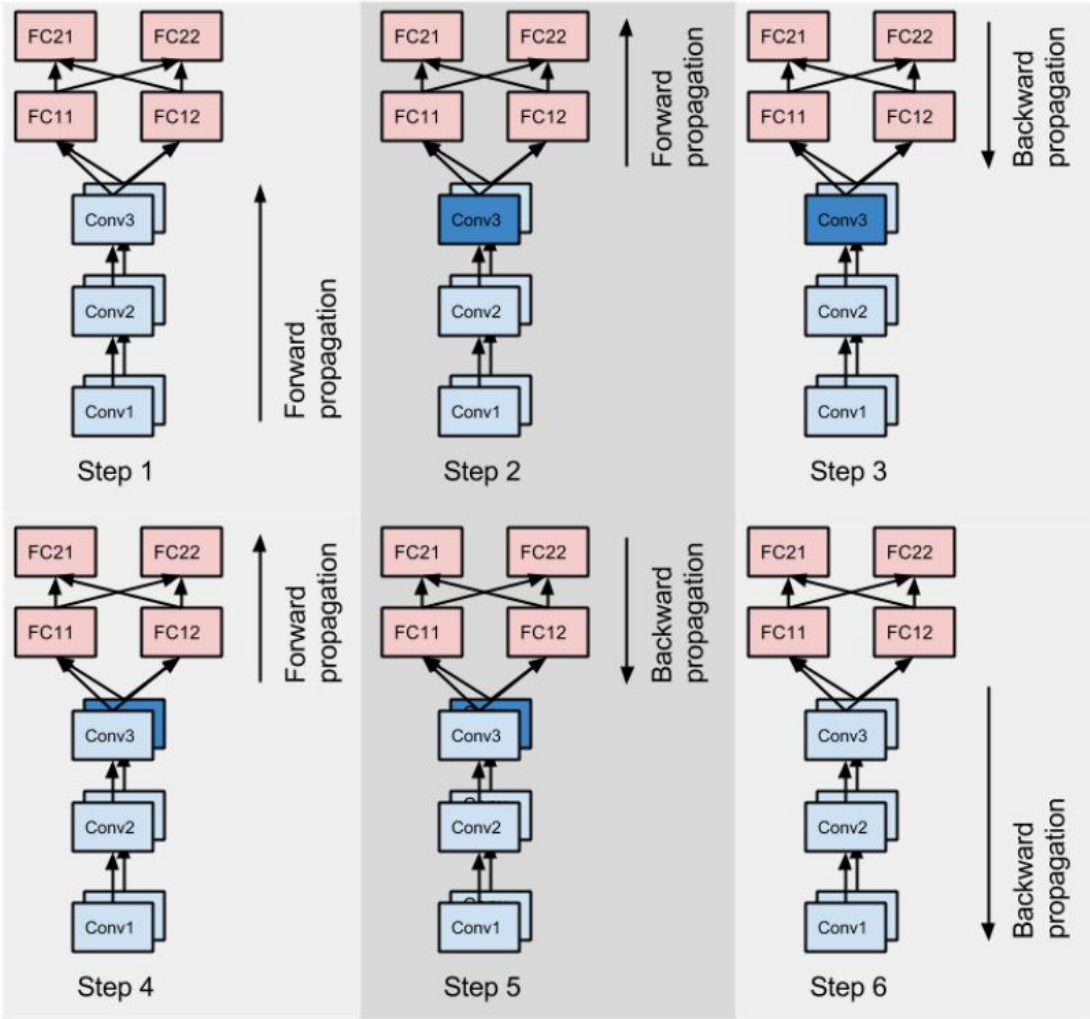


Fig: Illustration of the forward and backward propagations for scheme

MPI Communication Required calls

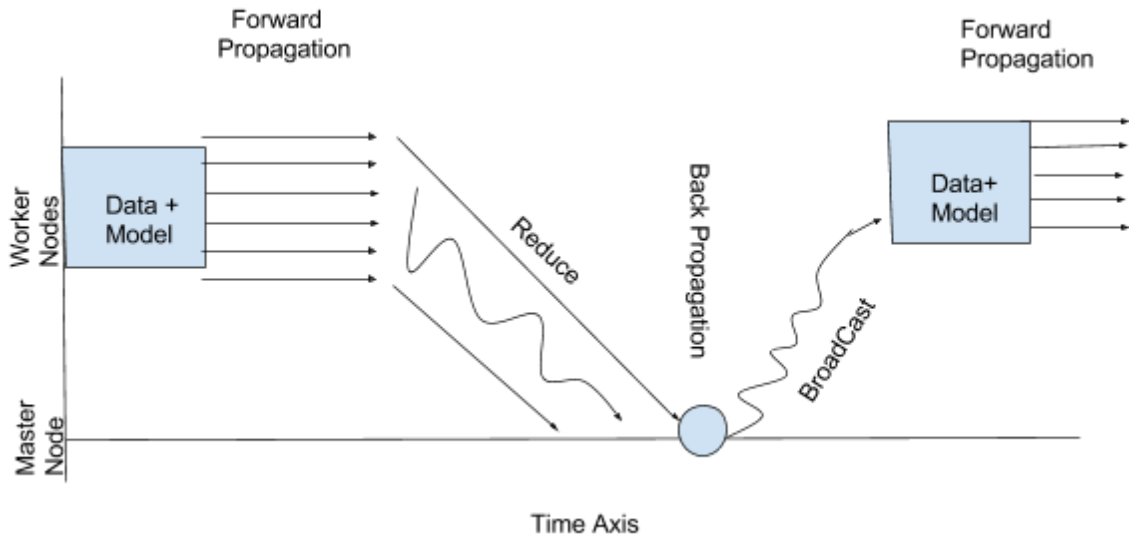
1) **Broadcast**
2) **Reduce**

Training a simple epoch of training data requires complete forward pass in training multiple layers (Convolution , pooling layers, fully connected neural layers) and in order to calculate the updates of the model for the given epoch , back propagation technique is used.

Reduce

For a data parallel model it is necessary for all the nodes to maintain the same model at any given point of time. Once forward pass is completed in all the nodes, a common aggregated error is calculated by using an "reduce" operation. Backpropagation as mentioned earlier is a sequential execution of updates in reverse order of the layers i.e. calculating the last layers

updates first and first layers updates last. Thus every worker would be idle during the phase of reduce operation and backpropagation. This forms the biggest bottleneck of the training time.



Figures showing nodes activities during forward propagation

Computation and Communication Profile

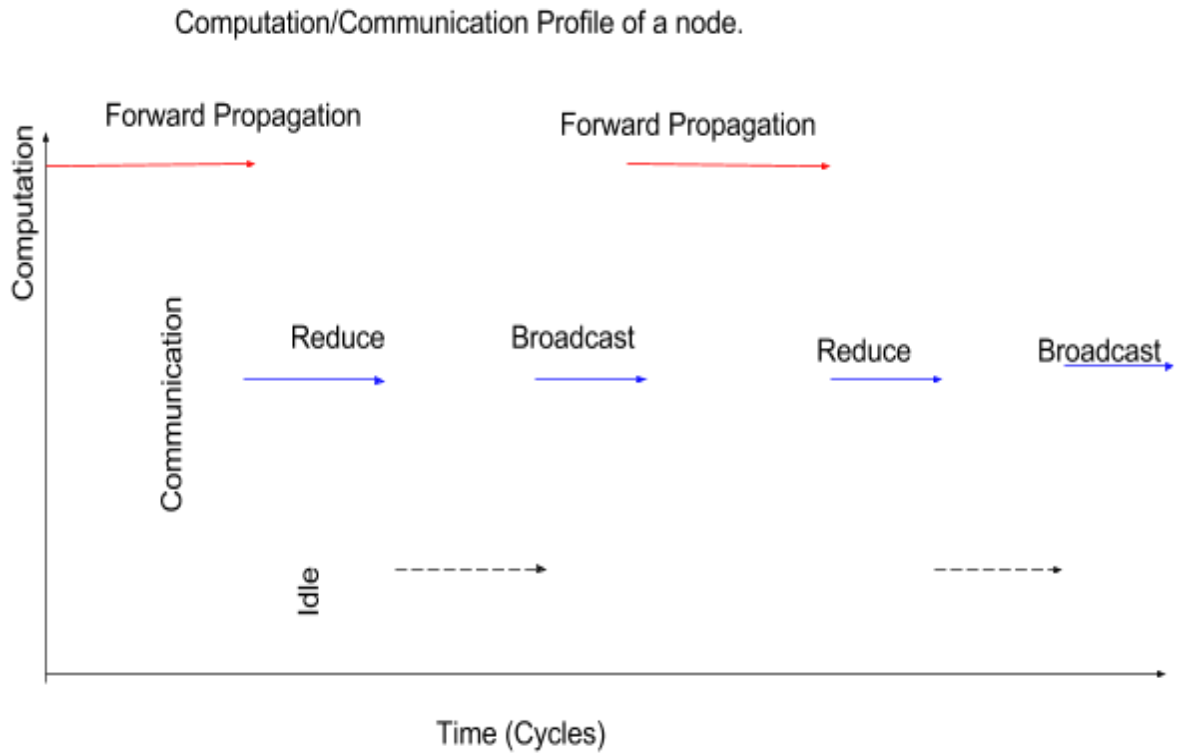


Figure showing Trend of Computation, Communication, Idle time profile for nodes

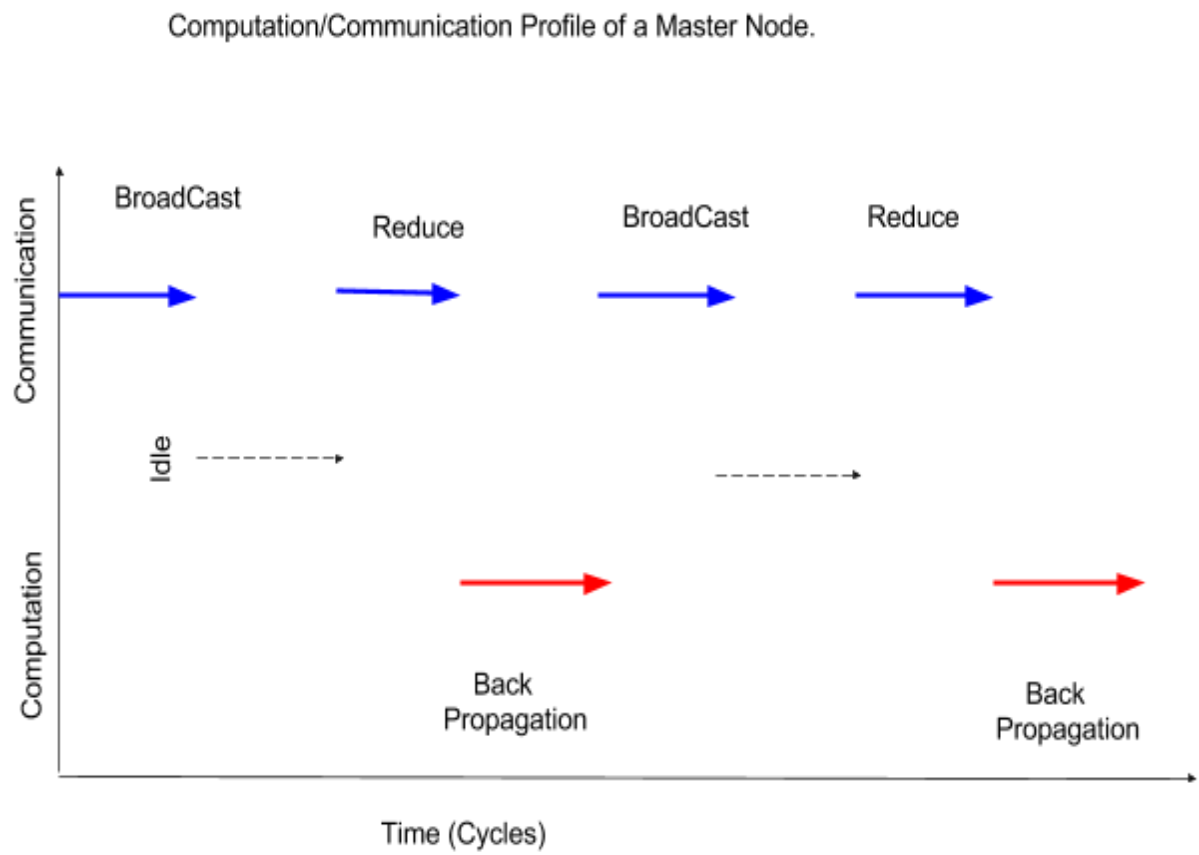


Figure showing Trend of Computation, Communication, Idle time profile for nodes

BroadCast

After establishing the final aggregated error , model is updated by propagating the error. The updated model needs to be broadcasted to all the nodes so that they can start training next set of training data.

In case the size of model is too large, broadcasting the entire model might lead to heavy congestion and increased latency. In order to overcome this, the model can be broadcasted in chunks of the model as forward propagation works in sequence of the layers . However it is required to maintain that the broadcast needs to follow the sequence of layers. Using this approach waiting time can be reduced and a better overlap of computation and communication is achieved.

Merits

1) Simple, efficient and effective parallel strategy
2) Least possible communication calls

    a) Method1:
    Communication messages in transit per Iteration = 2* Nodes
    Average message size = size(Model)

Payload of entire network traffic per iteration = 2*N*(Size(Model))

b) Method2:
Split the model into K chunks, in order to reduce the congestion.
Communication messages in transit per Iteration = Nodes(1+ K)
Average message size = size(Model / K)
Payload of entire network traffic per iter = 2*N*(Size(Model))
Payload of entire network traffic per iter per epoch = s*(size(Model))

3) Efficient usage of cpu cycles as back propagation is calculated only in one node and broadcasted to other nodes.

De-Merits

1) Does not work of models which are really big as each node needs to handle
2) Granularity in learning is lost.
3) Rate of Learning is slow as it can learn updates only after the completion of an iteration.
4) Reduce forms the biggest bottleneck of the training time.
5) Network congestion would be either too high or too low.

Model Parallelism

Model parallelism involves the strategy of training ConvNet by distributing the model across multiple nodes. However this is slightly complicated compared to data parallel, as the model needs to be synchronized at every layer during the training, the communication frequency and quantity of messages increases while size of messages is limited compared to data parallel method. The advantage of using this model is reduced memory demand on every node of the cluster as the entire model is shared among the cluster. This model would be greatly handful if the model size is too big. However when any trained models is used in production environment, it is more advantageous to use model parallel as compared to data parallel.

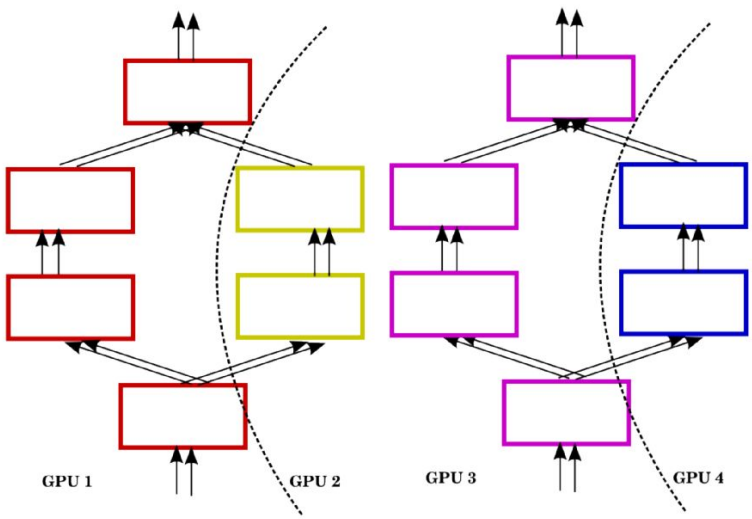
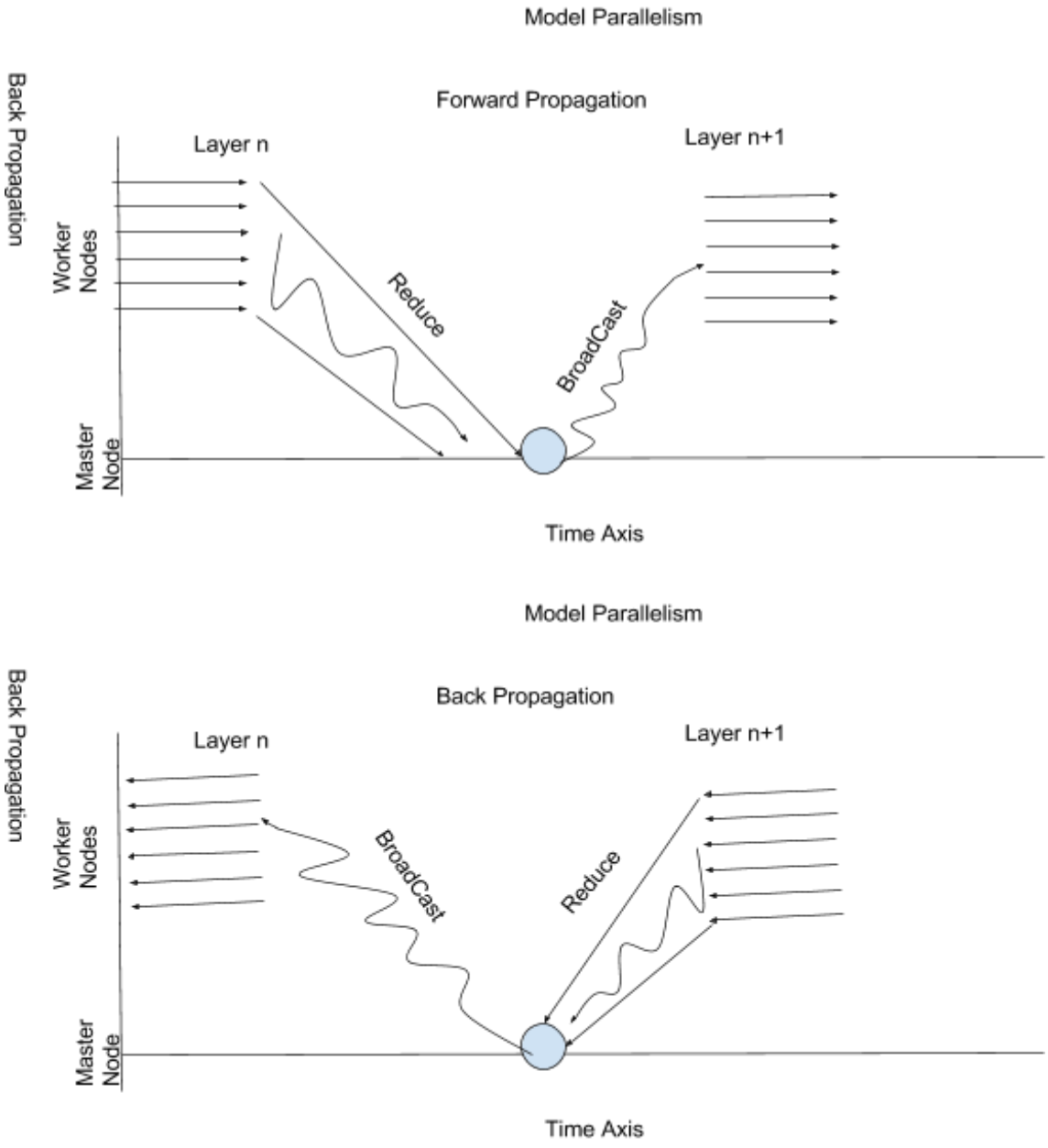


Fig: Example of how model and data parallelism can be combined in order to make effective use of 4 GPUs.

MPI Communication Required calls

1) **Broadcast**
2) **Reduce**



Figures showing nodes activities during forward and backward propagation

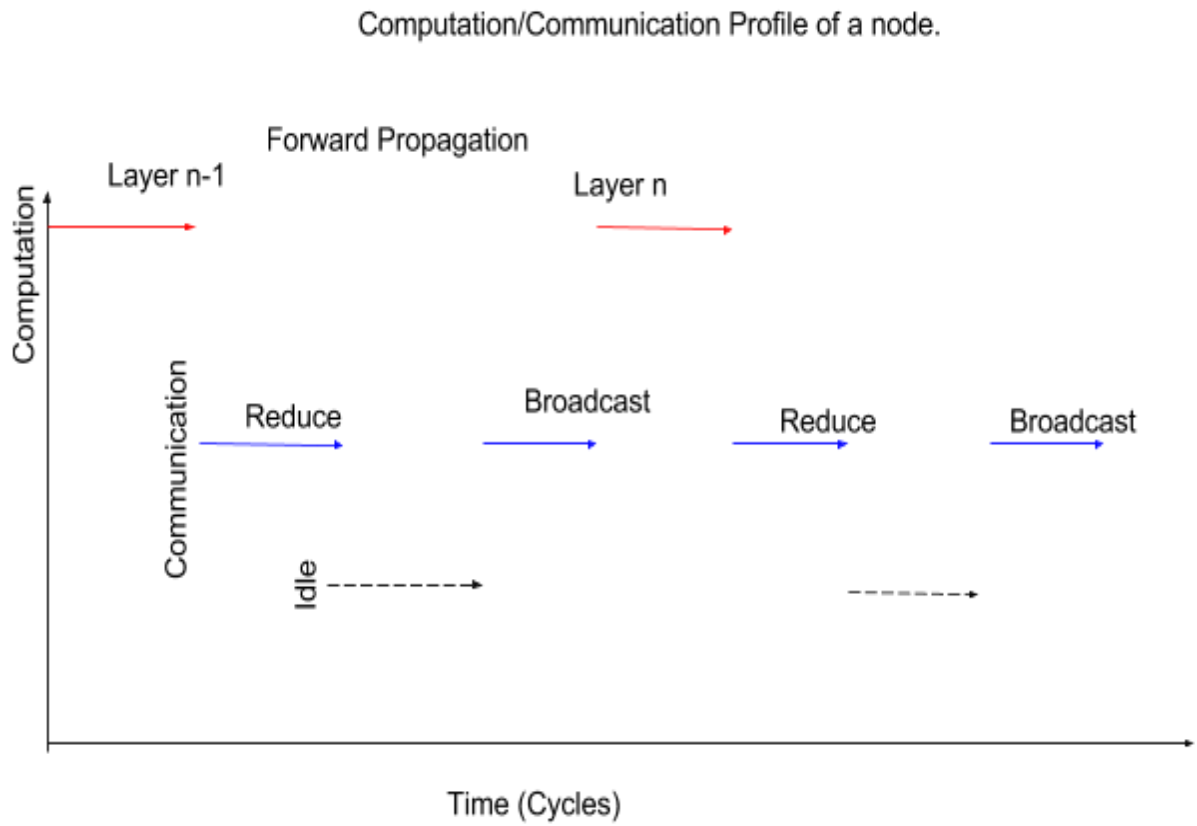


Figure showing Trend of Computation, Communication, Idle time profile for nodes

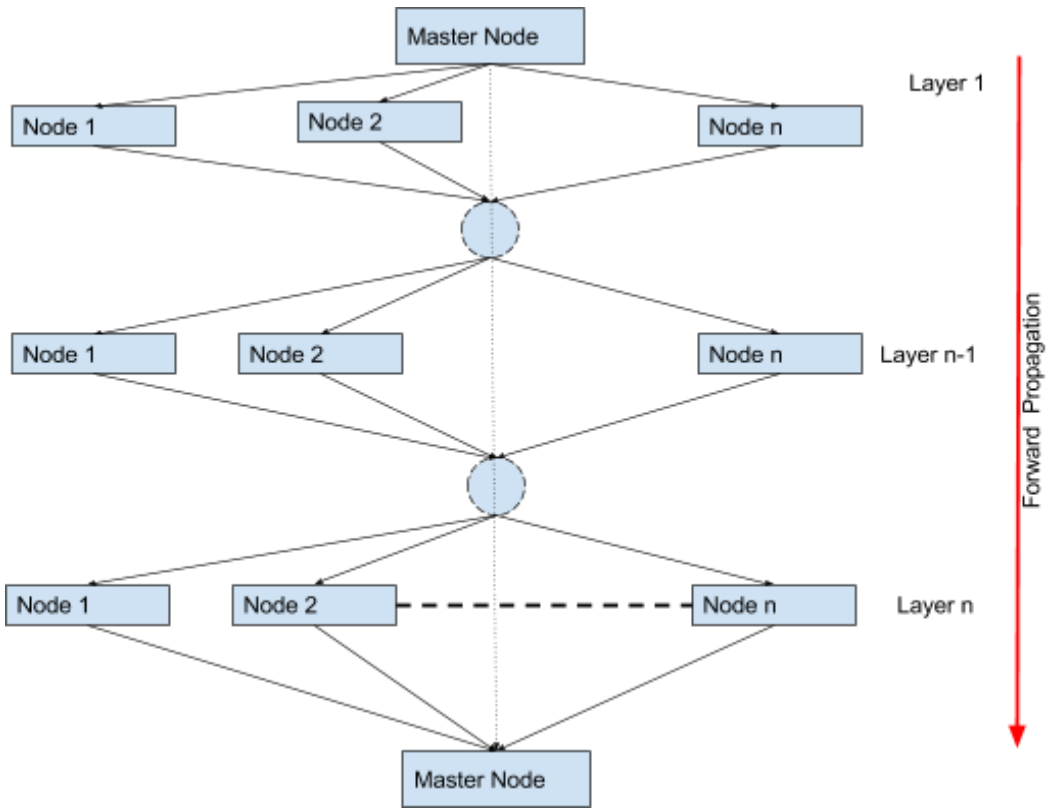


Figure showing the flow of data across nodes during forward propagation

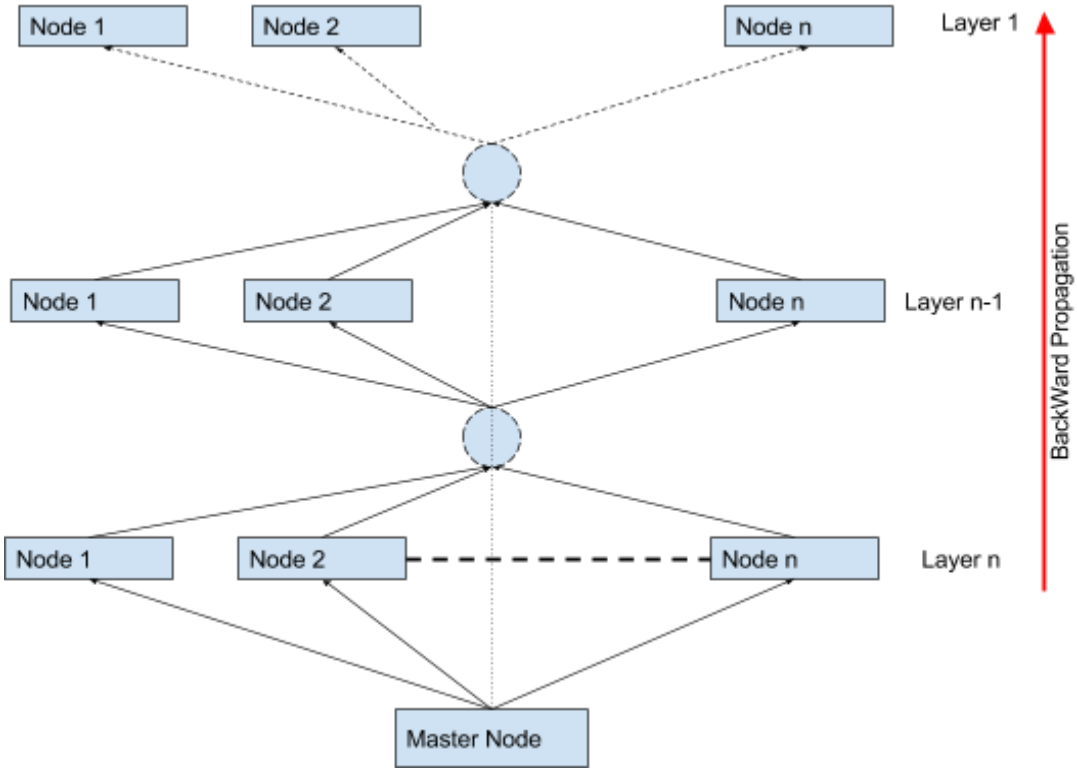
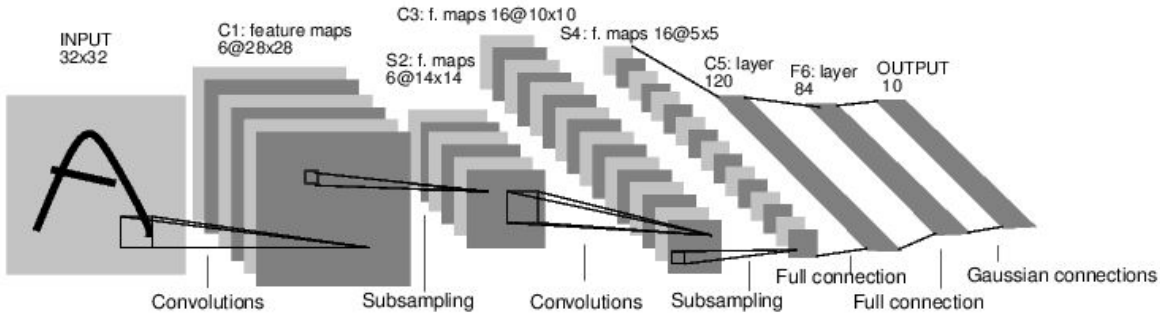


Figure showing the flow of data across nodes during backward propagation

Dynamism Behind Model Parallelism

Let's consider the following model(Lenet 5 By Alex) in order to understand how the model complexity grows in a deep learning network.



Types of Layers *Conv → → → → → → → Conv → FC → FC → GC*
Matrix Shapes *6*5*5 → → → → → → → 16*6*5*5 → 400*120 → 120*84 → 84*10*
Model Size *150 → → → → → → → 2400 → 48000 → 10080 → 840*
Data Shapes *32*32 → → → → → → → 6*14*14 → 16*5*5 → 120*1 → 84 *1 → 10*1*
Data Size *1024 → → → → → → → 1176 → 400 → 120 → 84 → 10*
Mode/Data *0.15 → → → → → → → 2.04 → 120 → 90 → 10*
Total Model Size = 61320

Total Data size = 1790

Conv → Convolution Layers FC → Fully Connected layers , GC → Gaussian Connections / Sigmoid Connections/ Logistic Regression etc...

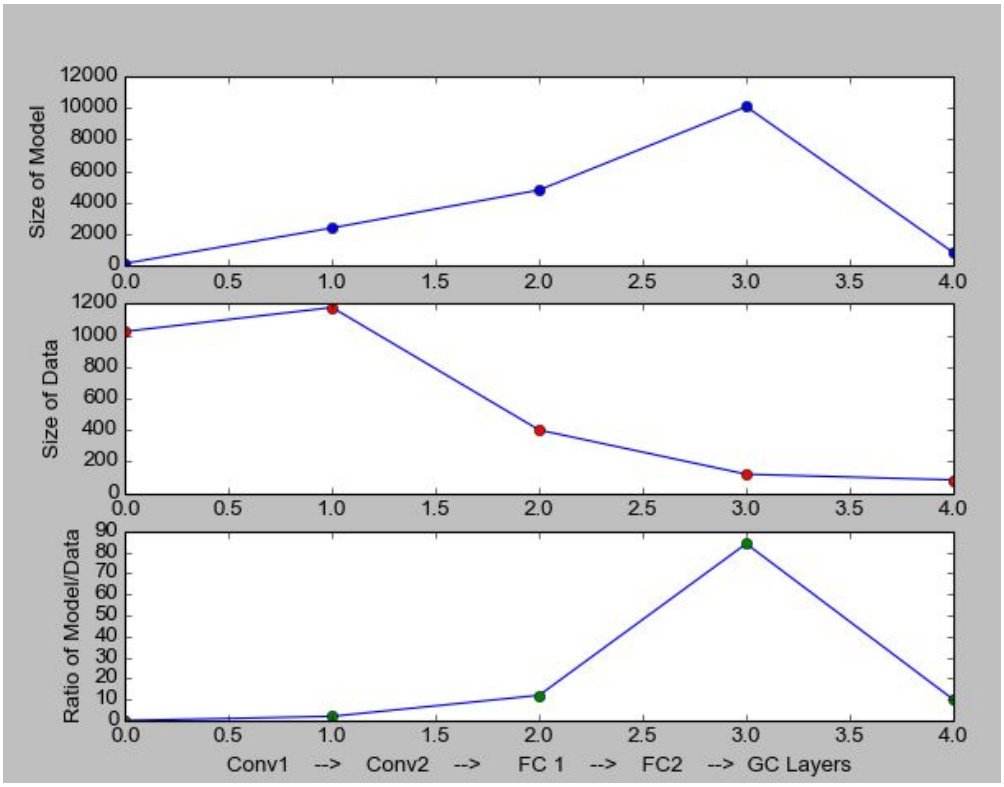


Figure showing sizes of data,model in various Layers of Deep Learning Architecture

In a model parallel architecture, we share the model against nodes and during forward and backward propagation, data seen by the next layer in forward propagation and error seen by the previous layer in backward propagation are reduced and broadcasted to all nodes. Since data is always far less than (at the most , two times of model) and reduce and broadcast are done per layer , maximum network traffic congestion would be only a small fraction compared to data parallel model leading to a uniform traffic in the network.

Merits

1) Relatively reduced congestion in the network and uniform distribution of the congestion throughout training.
2) Can handle deep learning model of any size and scale.
3) Fine grained learning is possible as model is distributed across nodes
4) Accelerated learning can be achieved using model parallelism.
5) Messages
   Communication messages in transit per Iteration = 2* Nodes* Layers
   Average message size = size(Total Data Per Iter / Layers)
   Payload of entire network traffic per iteration per epoch
   = 2*N* Layers* (Total Data Per Iter/ Layers)
   = 2*N* (Total Data Per Iter)

= 2*N*(size (Data Per Iter))

6) The entire network traffic is only a small fraction of the network traffic observed in data parallel.

Ratio = 2*N*Size(Data per Iter)/ 2*Size(Model)
= N * (1790/61320)
= 2.9% * N

For, Lenet 5 data set, it seems that for N <= 35 (Number of nodes) , the network traffic in Model parallelism would be less than that of data parallelism.

Demerits

1) Major disadvantage of the model parallelism is lack of fault tolerance. In case a node fails, the entire cluster can't forward until the node maintains a replica. Though some checkpointing can be maintained but comes with its own set of technical burden and debts.

2) Increased burden on programmers and with increasing complexity of the model, programmers life only gets harder.
3) Since communication/computation is very high and involve frequent communication,it could suffer from severe productivity loss due to network related issues like latency, high congestion , etc…

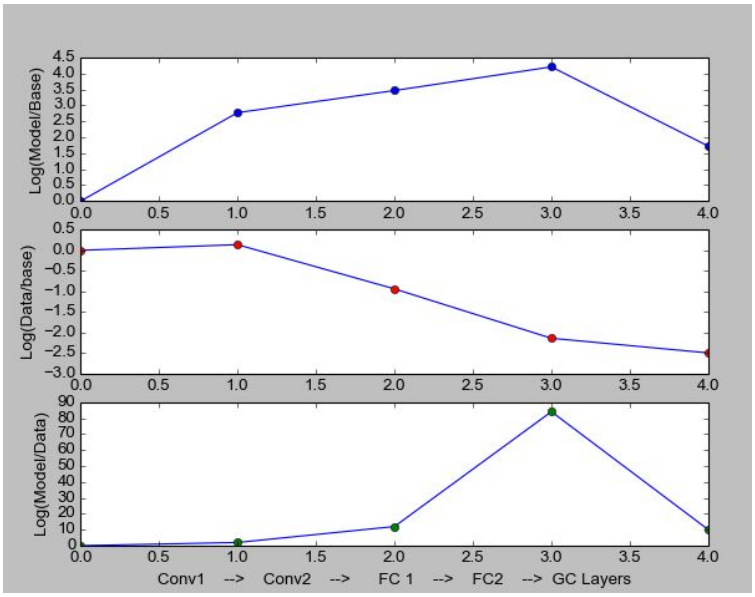


Figure showing relative sizes of data and model of various layers to initial Convolution layer1

Hybrid Model

The above graph explains a lot of dynamism in choosing model parallel over data parallel architecture. As we move from Convolution layers to fully connected layers, size of the model increases drastically and size of the data required becomes decreases. One could clearly see the ration of size of model to data clearly increasing. It is to be noted that fully connected layers model size is drastically big compared to the convolution layers model size. With increase in the size of the images, the model complexity grows polynomially and data parallel architecture might not be able to handle such cases due to memory limitations. One strategy to overcome this limitations is to use model parallelism for fully connected layers and data parallelism for convolutional layers.

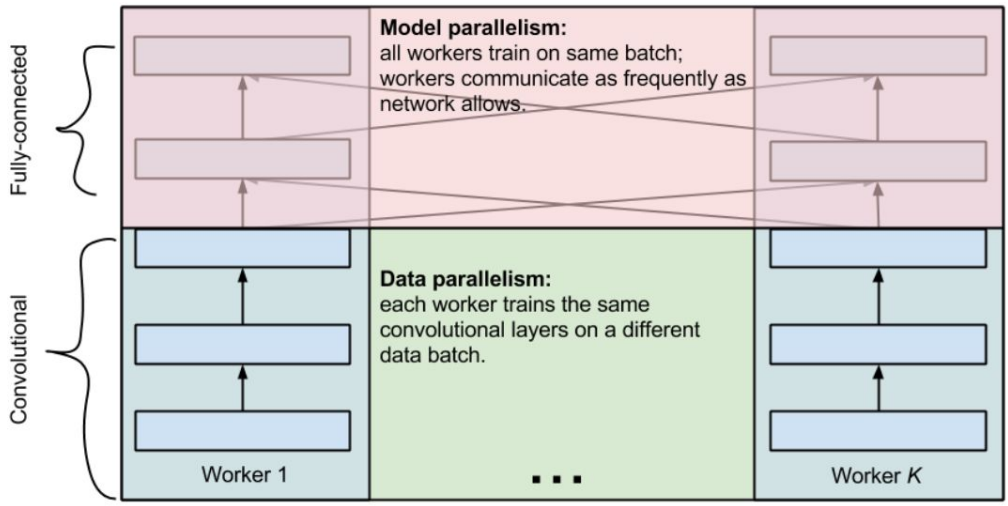


Fig : Difference between Model and Data parallel

Implementation models Considered

In order to test the three strategies namely conventional, data parallel and model parallel, three different clones of Berkeley open source Caffe namely

1) BVLC caffe (supports GPU but not MPI)
2) Data Parallel Caffe Model by Yixiong (supports GPU and MPI)
3) Model Parallel Caffe by Steflee (supports GPU and MPI)

Results

GPU Mode

S.NO	Mode (GPU / CPU)	Processes	Max Iterations	Batch size	Accuracy (%)	Time (seconds)	Time (min)	Completed
1	GPU	2	4000	10	42	17	0.28	yes
2	GPU	2	4000	50	66	30	0.5	yes
3	GPU	2	4000	100	70	42	0.7	yes
4	GPU	2	4000	200	71	81	1.35	yes
5	GPU	2	4000	400	74	150	2.5	yes
6	GPU	2	4000	800	76	315	5.25	yes

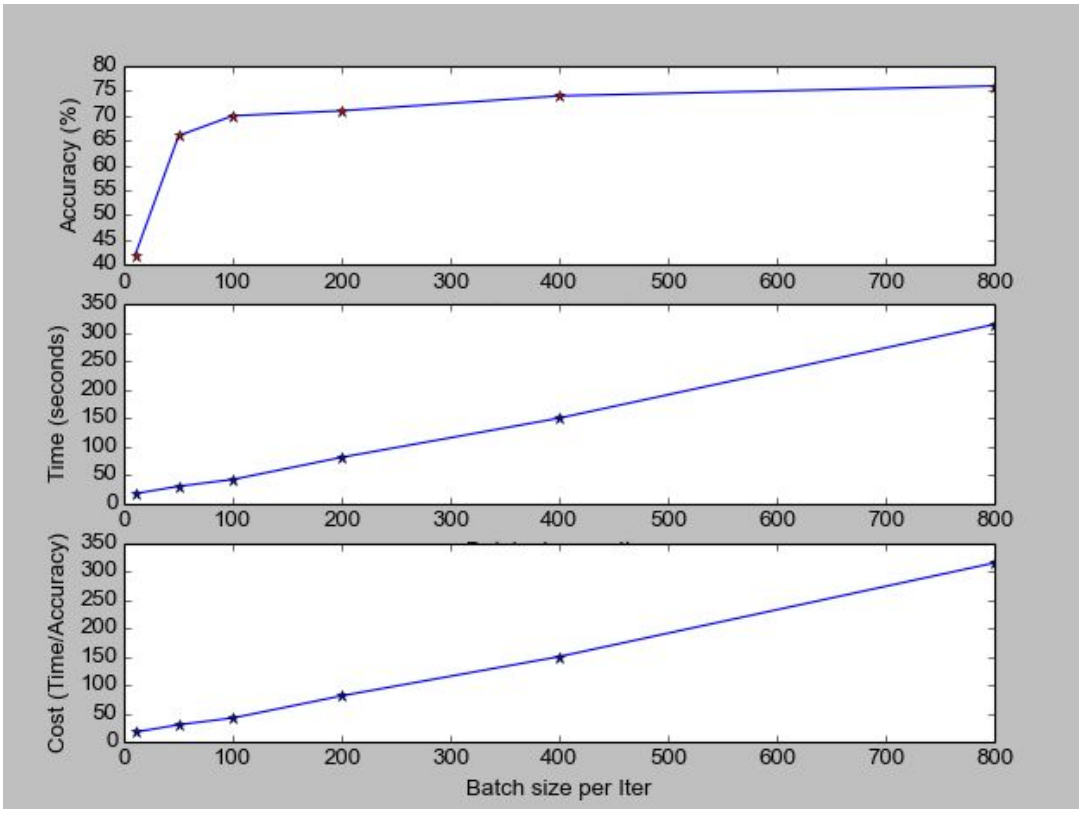


Figure showing Performance against various batch sizes.

Dry Run (CPU Mode)

S.NO	Mode (GPU / CPU)	Processes	Max Iterations	Batch size	Accuracy (%)	Time (seconds)	Time (min)	Completed
1	CPU	2	4000	10	43	106	1.77	yes
2	CPU	2	4000	50	69	483	8.05	yes
3	CPU	2	4000	100	72	1018	16.97	yes
4	CPU	2	4000	200	71	2018	33.63	yes
5	CPU	2	4000	400	74	5740	95.67	Failed

GPU Mode

S.NO	Mode (GPU / CPU)	Processes	Max Iterations	Batch size	Accuracy (%)	Time (seconds)	Time (min)	Completed
1	GPU	1	4000	200	71	75.042	1.25	yes
2	GPU	2	4000	200	74	75.71	1.26	yes
3	GPU	3	4000	200	74	81.68	1.36	yes
4	GPU	4	4000	200	70.85	82.67	1.38	yes

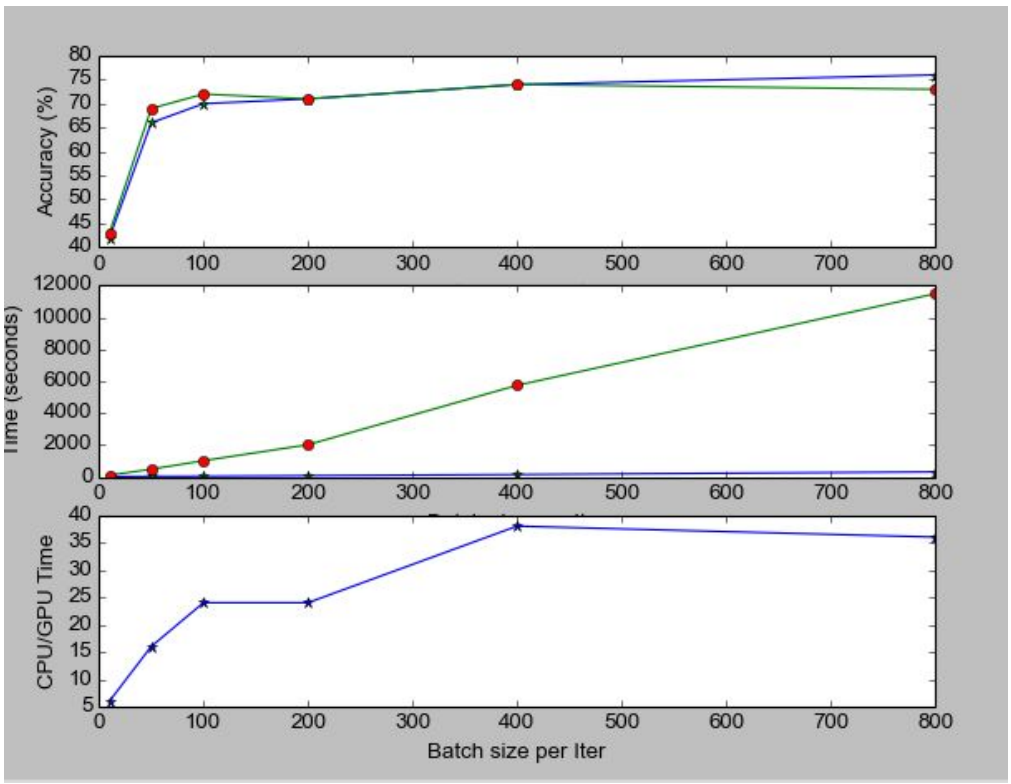


Figure showing CPU Vs GPU runs for varying batch sizes.

To be added

1) Verified Data parallel MPI results
2) Verified Model Parallel MPI results

Conclusions

1) 30x improvement computation time while using GPGPUs
2) Accuracy improved with increase in batch-size up to a certain point after which there wasn't any sign of improvement but rather there was a drop in efficiency.
3) While maintaining a fixed number of iterations, training time is linear to the batch size for a given configuration of no of processors and mode (GPU/CPU).

Shortcomings and Future work

The following shortcomings could be a great subject for any further work and if possible, would like to continue working on it.

1) Couldn't build a MPI - model parallel model as there wasn't any proper instructions / support from the community for a MPI -Model_parallel model.
2) Couldn't implement any new hybrid models (data Parallel + model Parallel)

References:

1) One weird trick for parallelizing convolutional neural networks by Alex Krizhevsky (Google Inc.)
2) Multi-GPU Training of ConvNets by Omry Yadan,Keith Adams,Yaniv Taigman,Marc'Aurelio Ranzato (Facebook)
3) ImageNet classification with deep convolutional neural networks. In NIPS, 2012 by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton.