# Acceleration of the Deep Learning Algorithms on different hardware platform

Sahil Kaw

Computer Engineering,

Charles Davidson College of Engineering, SJSU,

San Jose, USA

[Sahil.kaw@sjsu.edu](mailto:Sahil.kaw@sjsu.edu).

SourabhKetkale

Computer Engineering,

Charles Davidson College of Engineering, SJSU,

San Jose, USA

[Sourabh.ketkale@sjsu.edu](mailto:Sourabh.ketkale@sjsu.edu).

GouthamNekkalapu

Computer Engineering,

Charles Davidson College of Engineering, SJSU,

San Jose, USA

Goutham.Nekkalapu@sjsu.edu

***Abstract*— Deep learning algorithms have now become hugely popular in research community due to their ability to achieve higher accuracy than conventional machine learning process in many broader areas of research. Deep learning algorithms rely on multiple cascaded layers of non-linear processing units, typically composed of hidden artificial neural networks for feature extraction and transformation over a large period of training sessions on the machines and thus are computationally intensive resource demanding processes. Long learning time caused by its complex structure, however, limits its usage only in high-cost servers or many-core GPU platforms so far. There has been a paradigm shift on the research trends for accelerating the deep learning algorithms by improving the hardware acceleration aspects. Our paper focuses on accelerating Deep learning algorithms by speeding-up specific parameters of Deep learning algorithms and proposing techniques of improvising the computation power on different hardware platforms as Intel Xeon Phi, CPU, GPU, low platform devices and FPGA. Deep Learning is characterized with the parallelism of data which make use of Hadoop MapReduce framework augmented with Intel Xeon phi to accelerate Auto encoders and with FPGA, with support of Caffe library, in-order to accelerate the performance.**

# Introduction

The inherent parallelism of GPUs makes it highly suitable to train deep learning algorithms. So far, many

Siddhi Pai

Computer Engineering,

Charles Davidson College of Engineering, SJSU,

San Jose, USA

[Siddhi.pai@sjsu.edu](mailto:Siddhi.pai@sjsu.edu)

Prince Jacob Chandy

Computer Engineering,

Charles Davidson College of Engineering, SJSU,

San Jose, USA

[prince.jacobchandy@sjsu.edu](mailto:prince.jacobchandy@sjsu.edu)

approaches have been taken to redesign and optimize systems to make it more suitable for deep learning training experiments and most of them are either single GPU-based systems or CPU-based distributed systems. But the performance of the single GPU-based system is comparatively slow, so as to make use of the GPU-based distributed system which is highly efficient and powerful as it makes use parallelism achieved by the clusters of GPU’s along with a GPU-specialized parameter server collectively also known as GeePS.

While the current solution has been to make use of clusters of graphics processing units (GPU) as general purpose

processors for tackling giant, difficult machine learning tasks using neural networks, auto-encoders or Deep Belief Network(DBN). The requirement of the huge computational resources in-order to train multi-layered neural networks requires a clusters of GPU, but due to high cost, component reliability and programming complexity the researches have shifted their focus on other many low powered, economical, scalable and easily implemented hardware devices such as CPU, FPGA, XEON PHI and low platform devices like portable devices and mobiles.

CPU’s are easily accessible hardware’s and provides ease of computation and implementation. The only challenge is to attain the speed-up in proximity to the Basic Linear Algebra Subprogram (BLAS) packages for implementing arithmetic operations and GPU for the implementation of the Neural networks. We would discuss the limitation of the Floating point implementation of the CPU and show in particular that by implementing the Fixed Point arithmetic with SSE3/SSE4 instruction we attaining the speedup by factor of 3 from the optimized BLAS packages and by using further enhancements we were able to optimize the speedup for the speech recognition neural networks on the CPU. To take our analysis one step ahead and find an alternative to high computing GPU, we tried to optimize the performance of the Deep Learning on a multi-core processor XEON PHI which is a cluster of 60 cores of CPU and each core provides 4 hardware threads and an inbuilt memory of 8 GB. The use of the Math Kernel Libraries (MKL) provides optimized parallel implementation of the matrix operation but brings with it the problem of thread oversubscription which can be effectively reduced using the MapReduce methods.

Apart from increasing the computation power of the CPU by taking the above approaches there is something more which can be done to make use of CPU’s for Neural network implementation. We discuss this in details through Project Adam. Project Adam system comprises of a data server which provides the training data input, model training system which is the heart of the system where the model actually gets trained and a model parameter server, where ultimately the trained model is stored as combination of trained weights. Model parallelism and data parallelism is used to achieve scalability. Basically, multiple model replicas, each consisting of multiple machines, that train on different subset of data and they all publish updates to the global model parameter server. The key optimizations include whole system co-design and exploitation of asynchrony (weight updates are commutative and associative). Whole system co-design includes model partitioning and local weight computation at the parameter server. Asynchrony exploitation is achieved due to multi-threaded weight updates without locks and asynchronous batch updates.

The field programmable gate arrays (FPGA) provide an economical alternative in terms of power. The use of FPGA’s for the deep learning purposes have become more main stream because of the recent development and availability of design tools which have removed the hindrance of configuring FPGA with custom hardware. Another factor that makes FPGA more desirable is their ability to provide substantial high performance per watt of power consumed. This has encouraged researchers to carry their research to deploy and test deep learning algorithms on limited power resources scenario’s like smart phones, data Centre server stacks. Thus, this opens the whole new scope for research in accelerating deep learning algorithms on small devices with better throughput and power limited resources. With that been said there are two new approaches that could be followed to server the cause. 1) By exploiting the data parallelism by distributing the Deep Learning heavy computations of CNN involved in the deep learning onto the Hadoop clusters. 2) By introducing FPGA hardware module integration with the core processors to provide added acceleration to deep learning kernels. Since FPGA architectures are flexible, this could also allow researchers the ability to explore model-level optimizations beyond what is possible on fixed architectures such as GPUs.

Till now we have introduced Deep Learning algorithms on the high platform devices but with the implementation of the System on Chip (SoC) we are able to implement multiple parallelism to include the complicated functions of the Convolution Deep Belief Networks (CDBN) on the low platform portable devices such as mobiles etc. These convolutional operations run on the many core architectures because of its high computational throughput and memory bandwidth. Though these 1D and 2D convolution operations run on many core architectures but still being complicated, there is a bottle-neck which occurs on the registers rather on memory or cache and thereby making use of the register tiling we can enhance its performance. Whereas in multichannel 2D convolution operations we implement register tiling on high performance GPU’s and local memory tiling in Intel MIC.

# Approaches

1. CPU FLOATING POINT IMPLEMENTATION

The basic implementation of the CPU makes use of the Memory locality, Loop Unrolling and Parallel Accumulators, SIMD and Intel SSE2 for attaining its speedup. Memory locality means that once a particular location of the memory address is being hit then its corresponding nearby locations are also being loaded in the caches. So as to make use of the memory efficiently one should strive to make the most innermost loop of any numerical computation walk contiguously. Loop unrolling reduces the overhead of checking the loop termination and Parallel accumulators which provides access for pipelining the operations. Single Input and Multiple Data (SIMD) provides the fundamental blocks for low-level parallelization on CPU. The only limiting factor for SIMD instructions are that they perform much better on the 16 bytes block which are 16 byte aligned in memory. To make data 16 bit aligned, we perform zero padding. Streaming SIMD Extension 2(SSE2) provides the basic instructions to perform the multiply and add operations using floating point SIMD arithmetic.

1.1 COMPARISONS OF SPEEDUP

On comparing the matrix multiplication of the two 2D matrices making use of the Eigen(BLAS) packages, which are very fast libraries and provide cache optimization we found that baseline implementation appears faster for the thinner matrices but does not scale well for larger matrices.

Same for the comparison of the neural network implementation on the CPU and GPU we found that GPU were able to attain speedup by factor of 3 without batching

1.2 ENHANCEMENTS IN CPU PROCESS

The Fixed point implementation can be used to increase the speedup of the CPU by making use of certain properties of the Neural Networks(NN). The activations which are the outcomes of the neural network lie in the range of [0,1] interval which let them to be represented as the unsigned integers without scaling. The outputs of the primary layer of the NN are compressed through sigmoid signals (tanh). Quantization errors are subdued and do not cause instability because of the linear nature of the operations and dynamic range of the compression of the sigmoid function.

Linear Quantization

The input layer of NN accepts floating point in-order to accept the wide variety of inputs and 8-bit quantization techniques are used to convert activations to unsigned character and intermediate layer to signed char but biases are encoded as 32-bit int. This helps in reducing the memory footprint by 3 to 4 times of original. Though by this we are able to attain the speedup by factor of 2 from the floating point implementation but its still slower than SSE2 optimization technique.

Intel SSE3 and SSE4

The intel SSE3 make use of the pmaddubsw instructions which multiplies vertically each unsigned byte of the destination operand with the corresponding signed byte of the source operand and thereby provides high efficiency for the quantized neural network computation. Using these instructions, we are able to attain a speedup by factor of 3 and further SSE4 provided optimization for the conversion of the 16-bit to the 32-bit conversion. This further elevated the process speed by 9% over SSE3.

1.3 TASK SPECIFIC IMPROVEMENTS

Now using a neural network for achieving state-of-the-art performance on the task of speech recognition for mobile voice input we found that GPU outperforms the CPU. But by using certain optimization technique the CPU performance can be enhanced.

1.4 BATCHING

With batching we can improvise the memory locality and further increase the performance of the CPU. To take the advantage of the batching, the batches have to propagate through a neural network in bulk so that every linear computation becomes a matrix-matrix multiply which can take advantage of CPU caching of both weights and activations.

1.5 LAZY EVALUATION

During decoding of each frame of the speech it was found that only a fraction of the state square needs to be computed. As every state has its small set of Gaussians, only a fraction of the total parameter space has to be visited at every point. There are several well-known Gaussian selection techniques which further help narrow down the pool of Gaussians that need to be evaluated. In dense neural networks, every parameter has to be visited at every frame but the last layer, only needs to be computed for a given state if the state posterior is needed during decoding. This opens up the possibility of lazy evaluation whereby a state posterior is only computed when needed by the decoder.

1.6 BATCHED LAZY EVALUATION

Batched Lazy Evaluation make use of the both the batched and lazy evaluation to improvise the calculation speed of the neural networks.

2. PROJECT ADAM

Project Adam is an effort to build an efficient and scalable deep learning training system. Complex tasks like vision or understanding documents require large sized models to do a good job; model size grow linearly with the complexity of the task. As we have larger and larger models, to train the model to extract features automatically, linear amount of data is also required. Prodigious amount of computation is required to support such large models and huge amount of data. Large scale distributed systems are required to achieve this kind of computation.

2.1 MAIN IDEA

Project Adam system comprises of a data server which provides the training data input, model training system which is the heart of the system where the model actually gets trained and a model parameter server, where ultimately the trained model is stored as combination of trained weights. Model parallelism and data parallelism is used to achieve scalability. Basically, multiple model replicas, each consisting of multiple machines, that train on different subset of data and they all publish updates to the global model parameter server. The key optimizations include whole system co-design and exploitation of asynchrony (weight updates are commutative and associative). Whole system co-design includes model partitioning and local weight computation at the parameter server. Asynchrony exploitation is achieved due to multi-threaded weight updates without locks and asynchronous batch updates.

2.2 EVALUATION METHODS

Popular benchmarks for image recognition tasks, MNIST and ImageNet is used to evaluate Project Adam. MNIST is a digit classification task where the input data is composed of 28x28 imagesof the 10 handwritten digits [27]. ImageNet is a large dataset that contains over 15 million labeled high resolution images belonging to around 22,000 different categories [26]. We use this benchmark to characterize Adam’s performance and scaling, and the accuracy of trained models.Adam’s baseline performance is evaluated by focusing on single model training and parameter server machines. Then, baseline training accuracy is evaluated by training a small model on the MNIST digit classification task. We evaluate our system performance and scalability across multiple dimensions and evaluate its ability to train large DNNs for the ImageNet 22K classification task [25].

2.3 EVALUATION RESULTS

We found out that the model size and asynchrony do improve accuracy and performance of the system. Adam shows excellent scaling as we increase the number of cores since we allow parameters to be updated without locking. When the weight updates are computed locally we see good scaling as we have tiled the computation to efficiently use the processor cache avoiding the memory bandwidth bottleneck [25]. The model is able to achieve a new world record prediction accuracy of 29.8% using only ImageNet training data, which is a dramatic 2x improvement over the prior best. Adam achieves high multi-threaded scalability on a single machine by allowing threads to update local parameter weights without locks. It achieves good multi-machine scalability through minimizing communication traffic by performing the weight update computation on the parameter server machines and performing asynchronous batched updates to parameter values that take advantage of these updates being associative and commutative.

3. GEEPS

The inherent parallelism of GPUs makes it highly suitable to train deep learning algorithms. So far, many approaches have been taken to redesign and optimize systems to make it more suitable for deep learning training experiments and most of them are either single GPU-based systems or CPU-based distributed systems. Naturally, using GPU-based distributed systems should be an approach which should be experimented and this is what GeePS is about.

3.1 MAIN IDEA

This approach uses a cluster of GPUs along with a GPU-specialized parameter server to form the distributed system which support scalable data-parallel model training. The main idea is to overcome the disadvantages of previous GPU-based systems; inefficiency due to limited GPU memory size, GPU stalls and the data movement overhead between GPU and CPU. Here, data provided as input is divided across workers located in separate machines, which collectively update shared model parameters, which themselves are sharded across the machines. This way the delays due to communication and data movement can be avoided. Most of the processing can be done locally in the GPU as local-caches are introduced. The notable thing here is that the parameter server separates the problems of processing data and the problem of communicating and synchronizing them between different machines, taking care of the CPU to GPU memory data movement. To enable a parameter server to support parallel ML applications running on distributed GPUs the authors make three important changes: Explicit use of GPU memory for the parameter cache, Batch-based parameter access methods and Parameter server management of GPU memory on behalf of the application. The first two address performance, and the third expands the range of problem sizes that can be addressed with data-parallel execution on GPUs. The application still accesses everything through the GPU memory buffers, and the parameter server itself manages the movement of data between CPU and GPU.

3.2 EVALUATION METHODS

The goal of the evaluation is not comparing the classification accuracies of the various models, but on allowing faster training of the used model regardless of the model. The evaluation was done using 3 experiments, out of which two are image classification models and the other one is a video classification model. All the evaluations where executed using the original and minimally-modified Caffe application, which is an open-source single-GPU convolutional neural net application.

3.3 EVALUATION RESULTS

The evaluation shows that scaling is successful using GeePS, as a state-of-the-art single-node GPU implementation when used alongside GeePS scaled well, such as to 13 times the number of training images processed per second on 16 machines (relative to the original optimized single-node code). Moreover, GeePS achieves a higher training throughput with just four GPU machines than that a state-of-the-art CPU-only system achieves with 108 machines [5]. The results can be summarized as follows. GeePS provides effective data-parallel scaling of training throughput and training convergence rate, at least up to 16 machines with GPUs. Efficiency of GPS is much higher, for GPU-based training, than a traditional CPU-based parameter server and also much higher than parallel CPU-based training performance reported in the literature. Dynamic management of GPU memory in GeePS allows data-parallel GPU-based training on models that are much larger than used in state-of-the-art deep learning for image classification and video classification.

4. PROPOSED DESIGN FLOWS FOR DEEP LEARNING USING FPGA’S

The main architectural design challenge is to have a comparable compile time for compiling OpenCL kernels. The present FPGA’s by Xilinx and Altera involved takes around 10 minutes to hours, whereas compiling the Generic OpneCL kernels on the GPU counterpart takes time in milliseconds to seconds. This makes iterating through each design phase challenging with FPGA hardware if the compilation time is in hours. However, this is not completely futile as deep learning tools often reuse the precompiled kernels during the design phase. Most of the experiments carried out contain implementations similar designs using off-chip memory access, configurable software layers, buffered input-output, and other parallel processing elements implemented on FPGA fabric. Besides these the main crux lies in implementing efficient memory subsystems, data transfer mechanisms, interfacing library for processor core and FPGA.

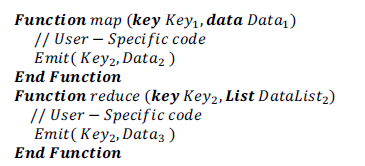
4.1 PARALLELISM IN DEEP LEARNING

Deep learning architecture, particularly CNN have shown tremendous outcomes in object classification, feature extraction, learning with the help of training through large datasets imparted over a period of time. Thus, proving the computation to get more demanding with increase in datasets to be fed for training. CNN training process is highly data parallel in nature. This parallel characteristic can be exploited by deploying them onto replicated models on different computational cluster nodes and by feeding partial data to train and compute weights individually. The computed weights can be aggregated over a central master server in the cluster. The proposed parallel distributed computing framework which suits well for the distributed deep learning is Hadoop clusters.

## The Hadoop MapReduce approach

The Hadoop map-reduce is a parallel programming model used to process large amounts of data on a distributed cluster of computing nodes. This model consists of 2 phases in data processing. First, a Mapper () function to process data from the database in (key1, value1) format and generate intermediate weight. Secondly, a reducer () function aggregates all these individual weights into a single weight which is again in the (key2, value2) format. To describe this into deep learning algorithms scenario when deployed on Hadoop, the input data is stored onto a compatiable distributed file system and is divided and fed to the master server. The master splits the input data and assigns each split to mapper node. Each node operated onto the incoming (key, value) pair data with the data category specific map () function on each node. Each node operates onto this key value pair data i.e this data is input to the Convolutional Neural network and Back propagation algorithm for feature extraction from the raw data. “The Backpropagation is a method to compute the gradient of the Objective function with respect to the weights and it is considered an application of the chain rule for derivatives. The

Derivative of the objective function with respect of the input to the modules is computed backward using the derivative with respect to the output of that module (beginning with the output of the whole network)” [28]. These outputs from CNN and back propagation algorithms are for aggregation to reducer functions in parallel. The resulting data from reducer function is averaged to form a final weight. This process is iterated on a large data inputs to train the Deep learning Algorithm. The generic mapreduce () function pseudo code is described as below [28]:



## Accelerating deep learning algorithm using FPGA hardware onto cluster node

A single Hadoop Node is added with a FPGA hardware which is architected to work along with the core processors. The prototype used in the experiment is equipped with xilinx 7000 FPGA all programmable SOC, with 2 ARM cotex-A9 processors. One processor serves the network requests in the cluster, while the other processor is configured with FPGA to process data over CNN, back propagation algorithms that run on FPGA depending on the rules set according to the distribution of the data processing implementation. The FPGA involves into speeding up of intensive computations involving multilayer network of non-linear transformations to be carried out on the data.

## Challenges involved [28]

1. The first challenge is to identify the performance bottlenecks in the Deep learning of CNN.

2. The second challenge involves the design and implementation of a Hadoop distributed architecture for the deep learning algorithm that takes into account the separation of kernels into modules that can be processed on either standard general purpose nodes or accelerated FPGA-based nodes. This is to allow datacenters to increase computation resources by adding FPGA nodes without having to replace existing general-purpose clusters.

3. The third challenge involves the design and synthesis of the reconfigurable architecture to support the desired kernel acceleration on FPGA.

4. The fourth challenge involves the development of the software library that supports the seamless integration

## Solutions [28]

* Kernel Identification:
* To identify the major performance bottleneck in CNN deep learning convolution operation using profiling tools and deploying those operations FPGA
* Approach to Distributed Algorithm with FPGA based nodes.
  + Each node runs parallel SGD method where the caffe library mappers to train the CNN deployed on each node. The caffelibray extends 2 extra classes to support the mapper and reducer function on the Hadoop node.
* Design and implementation of reconfigurable Architecture for deep learning
  + The communication latency between Core processor and FPGA is a critical parameter in acceleration. This was reduced by using a bus interface coupled with a DMA controller for communication.
* Seamless integration of Distributed Algorithm with the Accelerated Kernel
  + The main task is to check if the Hadoop node is a normal standard node or FPGA enabled. If FPGA enabled, then multiple thread work in parallel one to read input data and another to write the result, thus gaining speedup

5. LOW PLATFORM DEVICES

Cloud Computing is absolutely reliant on the Internet condition since it depends on associations between end-clients and main servers and sometimes results in security issue, data loss issue. As a remedy to these problems, another handling worldview called "Fog Computing" has come up [16]. Fog Computing is based on appropriated handling inside close end or far-end devices of big data framework. To enable deep learning and deep inference with Fog Computing, we have to reestablish the learning method because of dependency on the training data for smaller size of memory and also the efficient hardware has to be designed. Before developing a dedicated chip, we need to analyze and algorithm and hardware.

5.1 ANALYSIS OF DEEP LEARNING/INFERENCE ALGORITHM

1. *Deep-Layered Neural Network (DNN)*

Convolutional Deep Belief Network (CDBN) is a combination model of DBN and CNN. There are 3 hardware implementation issues of CDBN.

CDBN is based on probability based functions. CDBN has been trained based on ‘semi-supervised learning’ of CDBN where in it requires and addition stage of unsupervised learning. CDBN also needs additional hardware of Random Number Generator since CDBN requires randomly generated numbers.

1. *Tradeoffs in a Deep Learning/Inference Hardware*

In implementing a DL/DI hardware based on arithmetic precision, required performance, hardware parallelism, memory/IO bandwidth, and system of a random number generator, there are additional tradeoffs.

The block diagram of the DL/DI processor,

has a heterogeneous multicore architecture with

a 2D mesh NoC connecting 4 DL cores and 2 DI cores.

The TRNG (True Random Number Generator) is connected to the deep network learning engine (DNLE) and the deep network inference engine (DNIE) via an independent communication path. DL/DI processor has 3 clusters: a) DL core cluster b) DI core cluster and c) TRNG cluster. DL core cluster and DI core cluster share a main path for DL/DI

processing and are connected with TRNG through independent communication path to receive random numbers from TRNG to avoid the bandwidth reduction. DNLE consists of 4 DL cores where in each core has 4 dual-threaded task-level pipelined datapaths (DTPDs) inside for scalability. DNIE consists of 2 DI cores where in each core has 2 fine-grained pipelined per-cycle datapaths (FPPDs) inside for scalability. TRNG performs 16 -bit random number generation.

5.2 SYSTEM IMPLEMENTATION AND ANALYSIS

*A. Implementation*

The deep learning/ deep inference processor which gives high performance is made using 65 nm 8-metal CMPS technology, integrating 3.75M equivalent gates and 216 KB of SRAM for battery-powered personal devices. It gives us 42.1 X faster operation in CPU and 1.3 X faster operation in GPU and also consumes the 213.1 mW peak power when running at 200 MHz with 1.2 V supply voltage. With 11.3 GOPS peak performance, the deep Learning / inference processor achieves 1.93TOPS/W power efficiency-85.6 % improvement over a state-of-the-art on-chip deep-learning processor, [15] and its high scalability enables multi-chip implementation to realize real-time learning and inference for deep neural networks.

*B. Analysis*

So we present an energy-efficient and scalable DI / DI processor. DL with high energy-efficiency gets proved by 3 features: a) DNLE with a dual-threaded 4-stage task-level pipeline, b) DNIE with a dynamically reconfigurable systolic PE-array (DRSA), c) TRNG (True Random Number Generator) and dual layered architecture (DLA) with a meta stable entropy source (ES). DNLE increases DL processing by exploiting task-level and layer-level parallelism. DNIE reduces the SRAM bandwidth and enables per cycle inference by exploiting both fine-grained parallelism and neuron-level parallelism.

6. MANY-CORE ARCHITECTURES

Convolution operations used in application domains such as deep learning and computer vision is the most time consuming part. So memory bandwidth and high computational throughput make many core architectures the reliable targets.

6.1 1D AND 2D CONVOLUTIONS

*A. Basics of 1D and 2D convolutions*

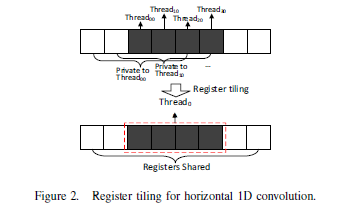
Gauss filter is used in 1D and 2D convolutions. For 1D convolution, convolution filter is a 1-dimension structure, consisting of horizontal filer of 1\*N and vertical filter of N\*1. For 2D convolution, the convolution filter is a 2-dimension structure, namely the filter size is N\*N.

*B. 1D and 2D convolutions on many-core architectures*

Implementing 1D and 2D convolution based on openCL which arranges threads in N Dimensional space. NDRange(work-item) = number of pixel of the output image, and each thread is responsible for one module. But thread will access global memory n times in this process; n is filter size.

We see overlap between data which is accessed by threads in 1D and 2D convolution [14].

This is inter-module data reuse. We can optimize it by using local memory tiling. It has lower latency and high bandwidth compared to global memory. Global memory access can be reduced n (filter size) times by local memory tiling. One more bottleneck is on the register file, where threads can’t see data of other threads. The same data will have many copies in register file since there are overlaps between adjacent threads.



Register tiling by combing the workload of each thread is used, for register data reuse.

6.2 MULTICHANNEL 2D CONVOLUTION OF CONVOLUTION NEURAL NETWORK

*A. Basics of multi-channel 2D convolution*

CNN’s convolution layer is a multichannel 2D convolution. Each filter has to convolute on each image for the serial algorithm, which forms the outermost 2-level for loops, looping on NumImages and NumFilters, respectively.

CNN’s convolution layer contains 7 nested for loops, in serial algorithm.

*B. A solution for multi-channel 2D convolution on manycore architectures*

To partition the tasks and set NDRange is important in implementing the convolution layer of CNN in OpenCL. A method is to make threads compute for one pixel of output images. In the OpenCL kernel, 3-level for loops are there, iterating on Channels, FilterSize and FilterSize, respectively. This is a direct translation from the serial algorithm.

NDRange is set as shown below:

local\_size[2] = {BlckSizeX, BlckSizeY};

global\_size[2] = {Width\*NumFilters/NumFltrsPerThrd,

Height\*NumImages/NumImgsPerThrd};

*C. Parameters for autotuning*

We can search the parameter space and find out optimal combination of parameters by autotuning.

For multichannel 2D convolution, is carried out on images directly and data reuse algorithm is exploited. Our solution there by reduces memory consumption and improvises data reuse as compared to CAFFE and other deep learning libraries. For 1D / 2D convolutions, register tiling is used to achieve high performance on both GPU and Intel MIC many core architectures. For 2D convolution, local tiling is used in Intel MIC and register tiling is used for GPU’s. Our solution gives up to 25% of the theoretical peak performance.

7. XEON PHI IMPLEMENTATION FOR NEURAL NETWORKS

The XEON PHI is a cluster of 60 cores of CPU and each core supports four hardware threads, a total of 240 hardware threads. Its is a powerful computing card having 8GB of memory and clock speed of 1 GHZ and thereby can be alternative to GPU for training large scale deep learning models on a single machine. Auto encoders are an artificial neural network and are the building blocks for Deep Neural networks. We construct the Deep Neural network by stacking the building blocks. The study applies the Intel Xeon Phi coprocessor to accelerate pattern matching in neuromorphic text recognition. Deep Neural Network is trained using XEON PHI processor. We make use of the Intel MKL (Math Kernel Libraries) in-order to attain parallelism in the process and control the number of threads to be given to the process. But still its observed that its difficult to set appropriate number of threads to every matrix multiplication and results in thread over subscription problem which degrades the performance of Xeon Phi. So we propose a map-reduce implementation of the auto encoder on Xeon Phi coprocessor and parallelize multiple encoder model with Bulk synchronous parallel (BSP) communication where the parameters are updated after computation of all the replicas.

7.1 ANALYZING AUTOENCODERS ON MATRIX OPERATION

The process of training Autoencoder is making use of the stochastic gradient descent(SGD) which is used to train Machine Learning models with large dataset. We make use of the mini-batches SGD algorithm to train the neural networks using small samples at a time and computing them independently.

To fully utilize the computing resource of many-core platform, the matrix operation is implemented using optimized linear algebra library such as OpenBLAS, the Intel MKL. Though we can achieve parallelism with MKL libraries but if an algorithm is performing several Matrix operations then its found that the performance of the operation using the single thread sequentially is 10 times better than concurrent operations and the reason for this is thread oversubscription.

Thread oversubscription results in the context switching and the cost of context switching on a many-core processor is expensive than it is on a multi-core, since the Xeon Phi core contains more register states.

The solution to this problem is to determine a way to control the number of the threads in programming. Deep learning model contains multi-layers with different number of neurons and task parallelism, it is very difficult to set appropriate threads to every MKL function. The map reduce function is the suitable way to meet this thread control requirement which is a popular distributed and parallel programming framework.

7.2 DESIGNING OF THE PARALLEL AUTOENCODERS ON XEON PHI

Generally, the CPU memory is used to store the training dataset, parameters and temporary variables for designing machine learning platforms in-order to avoid the I/O bottleneck. But due to the larger dataset of the training model we are not able to store everything in the global memory of the CPU thereby an extra time is required for the transfer of the data from host RAM and the memory. Thus this additional transfer time needs to be taken into consideration. But this whole data is not required in every iteration of the process so we keep all parameters and temporary variables always stored in the global memory of the Xeon Phi during training time, and only transfer a large block of training dataset.

Along with this Xeon Phi also supports Task and data parallelism. In mini batch Standard Gradient Descent (SGD) we process many samples to obtain the final average weight updates so thereby many model replicas are processed in parallel and computing a model replica is called as the task. The two constraints which arrives due to parallelism is that maximum number of the model replicas cannot exceed the total number of the threads and the other is that each replica is provided enough of the computational workload.

The first constraint is taken care by the user defined code by setting the number of threads using MKL libraries and the second constraint is taken care by using adaptive method to distribute workload dynamically.

We obtain Data Parallelism in the Xeon Phi by using the Vector Process Unit to compute data wise operation in each model replica. Along with the parallelism there should be mapping between the thread and the core. We determine this mapping using the affinity mode. Xeon Phi supports three affinity modes which are Compact mode, scatter and balanced mode. The Xeon Phi make use of the static of dynamic schedule mode to bind the thread to the core. From our experiments it was observed that balanced mode was much better than other two.

7.3 EVALUATION

1) Speedup on different model sizes: The performance was evaluated on different model sizes by varying the number of the neurons in the encoder layer. The matrix method was implemented using the MKL libraries and the other by using the map-reduce programming framework. The observation was made on a single CPU code and it was found for highermodels MKL provides an edge over the map-reduce method because we were able to make use of the all the threads of Xeon Phi while in smaller models map-reduce method was efficient.

2) Speedup on different number of model replicas: Different model sizes and training samples were taken into consideration and with increase in the number of the model replicas the speedup initially increases and then decreases. So thereby more number of models cannot bring speedup because of context switching but we can provide more number of samples to increase the computational workload of every replica.

From all the above evaluation it was found that effective parallel design of the auto encoders based on map-reduce programming framework can accelerate the auto encoder training.

8. THE POTENTIAL OF INTEL XEON PHIFOR

SUPERVIDED DEEP LEARNING

Deep Learning algorithms are becoming a core component of many modern applications including: self-driving cars, classification of liver diseases, and speech recognition [20]. Supervised learning of Convolutional Neural Net-works (CNNs), also known as supervised Deep Learning, is acomputationally demanding process. For training CNNs several weeks are required to complete a training session if

performed sequentially on a CPU. Compared to other devices (such as GPUs) used for acceleration of computationally intensive tasks, Intel Xeon Phi, deserves our attention because of programmability and portability. The architecture of Intel Xeon Phi is already discussed in section 8.

The authors propose an approach called “Controlled Hogwild with Arbitrary Order of Synchronization” (CHAOS) that is optimized for the Intel Xeon Phi co-processor. Thread parallelism is used to divide the input over the available threads, allowing threads to process samples concurrently. We apply SIMD parallelism in convolutional layers to the computations of partial derivatives and weight gradients.

8.1 CHAOS APPROACH

As mentioned above CHAOS approach improves the performance by thread parallelism. Also it tries to exploit the many core architecture of Xeon Phi. The key aspects of CHAOS can be described as:

a) Thread Level Parallelism

b) Controlled Hogwild

c) Arbitrary Order of Synchronization

d) Vectorization.

Updates of weight parameters in back-propagation are not instant nor significantly delayed [22].To avoid unnecessary invalidation of cache lines and align

memory writes, updates of shared weights are delayed to the

end of each layer’s computations. Intermediate updates are

done to local weight parameters, thus calculating the gradients before sharing them with other workers. This approach is termed as controlled Hogwild. In arbitrary Order of Synchronization, because all workers share weight parameters, there is no need for explicit synchronization. However, an implicit synchronization is done in an arbitrary order because writes are performed according to a first-come-first schedule and reads are performed on demand. Vectorization scheme allows to add SIMD parallelismto the computations in convolutional layers, aligned the memory allocations and memory access to 64 byte. SIMD parallelism in convolutional layers is applied to the computations of

partial derivatives and weight gradients, allowing for efficient

use of the vector processing unit.

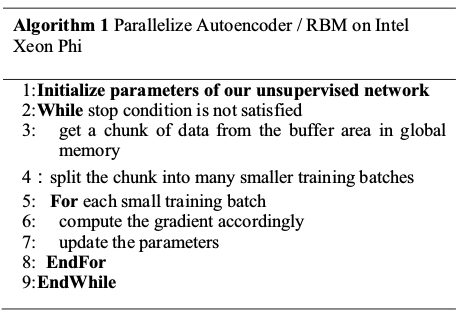
8.2 Results

For evaluating the performance of above proposed

scheme. The algorithm is implemented in C++ using OpenMP

to exploit thread- and data-parallelism. The application was

compiled natively for the coprocessor using the Intel compiler

15.0.0 and the O3 optimization option. All measurements were

carried out multiple times, and averaged.

Results presented for 244 threads show that the Xeon Phi yield a 103.5x, 99.9x and 100.4x speed up for the large, medium and small architecture respectively, when compared to one thread on the Xeon Phi coprocessor. Also the large CNN architecture, trained for 15 epochs, completes in 31.1 hours for the Xeon E5 (normal CPU without co-processors) and 2.9 hours using 244 threads on the Xeon Phi.

9. TRAINING LARGE SCALE DEEP NEURAL NETWORKS ON INTEL XEON PHI

In Deep learning unsupervised pre-training step allows us to find high-dimensional representations or abstract features which work much better than the principal component analysis (PCA) method. However, it will face problems when being applied to deal with large scale data due to its intensive computation from many levels of training process against large scale data. The sequential deep learning algorithms usually cannot finish the computation in an acceptable time. The authors of this paper propose a many-core algorithm which is based on a parallel method and is used in the Intel Xeon Phi many-core systems to speed up the unsupervised training process of Sparse auto-encoder and Restricted Boltzmann Machine (RBM)[23,24]. It achieves faster speed with better parallelism than the Intel Xeon CPU, motivation to use Intel Xeon Phi when compared to GPUs would be for better portability.

The unsupervised pre-training process of a deep neural network consists of many layers of unsupervised learning processes. The authors have proposed a parallel algorithm, with the intention of exploiting the architecture of Xeon Phi. First, memory transfers between the host and the Intel Xeon Phi is relatively slow. Thus, the number of data transfers between the host and Intel Xeon Phi should be minimized as much as possible. Therefore, we load the training data into the global memory of Intel Xeon Phi in a large chunk. Second, we use a thread to load the data chunk from the host to the Intel Xeon Phi so that our algorithm does not need to wait for loading new data when finishing the process of training one large chunk of data. This is a key point to keep all cores busy all the time.

The parallelized algorithm is described as bellow:

Among the steps mentioned in the above algorithm, computing the gradient is the most time-consuming for both Sparse auto-encoder and RBM. To parallelize the computing gradient step of RBM and Sparse auto-encoder, OpenMP and Intel MKL packages are used.

9.2. PERFORMANCE Evaluation

The algorithm was run on both the Intel Xeon Phi

platform and Single Xeon CPU core. The Intel Xeon Phi platform used comes with Xeon Phi 5110p many-core coprocessor. It is equipped with 60 active cores, each core with a frequency of 1.053 GHz, memory bandwidth of 320 GB/s and global memory of 8GB. The CPU we used to do experiments is Intel Xeon E5620, with frequency of 2.4GHz and 4 cores, and cache size of 12288 KB. When all cores and threads of Xeon Phi are used a speedup of 302x is obtained when compared to that of normal sequential execution.

# DISCUSSION

There is no need to come up with new machine learning or Deep Learning Algorithms to increase the efficiency, if there can be some powerful enough system with high computation environment. GPU’s has been an older choice for the researchers because of its property of conducting the operations in parallel. But still the speedup of the single core GPU is not so fast. So as to improvise upon the speedup we implemented clusters of GPU known as GeePS. Minimized overhead by overlapping the transfers with the training computation, without the two interfere with one another is the advantage of the GeePS over single GPU. Taking into consideration the high power requirements, component reliability and the programming complexity of the GPU other hardware devices like FPGA, CPU and Low Platform devices were made use of. CPU as a single core processor can though attain a speedup from its baseline floating point implementation and BLAS packages by making use of certain implementations but still it’s not very efficient for a very high calculative neural processes. We can overcome it by using clusters of CPU but still have to tradeoff with the power and other factors. But the availability of the multi core processors like XEON PHI is an advantage to perform such complex neural network calculations. It comprises of 60 cores of CPU having 4 hardware threads each and thereby 240 threads. XEON PHI has an advantage of the easy code portability. The limitation of the Xeon Phi is if the number of parallel processes performed on Neural network should not exceeds the number of parallel threads support by it. On the other hand distributing the deep learning algorithms on Hadoop cluster augmented with FPGA computing hardware has showed remarkable speedups for the computation intensive CNN and back-propagations algorithms in deep learning. FPGA’s have also showed reduced power consumptions per watt for which has made them favorable for resource limited ecosystems. And with support for open source libraries like caffe, torch for FPGA, researcher have started to consider FPGA’s over other hardware options.

# FUTURE RESEARCH DIRECTION

* A system which uses model parallelism to partition work across GPUs, rather than the simpler data-parallel model used in GeePS or even a combination of model parallelism and data parallelism and to reduce the involvement of CPU processing as well as the data movement between CPU and GPU.
* Hadoop clusters with automatic reconfigurable hardware accelerators on like FPGA by enhancing the caffe library functions and completely abstract the hardware complexeties from the computer vison library users.
* Improve the performance of multichannel 2D convolution operations of deep learning algorithms for small ﬁlter size, like 3\*3 and 5\*5.

# Conclusion

In this survey paper we have tried to study various approaches and techniques to accelerate the deep learning algorithms. We have found that though GPUs because of their inherent parallel architecture seems to be the best option, there are also options like FPGAs, Intel Xeon Phi. Also by some changes to the algorithms to make it more parallel by using the threads and many cores and techniques like controlled hogwild, batching, lazy evaluation we can achieve significant speedup when compared to normal sequential operations.

If our applications require low power consumption, then we can go for FPGAs rather than GPUs. Similarly, by choosing Intel Xeon Phi processor because of its programmability and portability. It takes less effort to port to this platform when compared to porting to GPU platform. By introducing techniques like HADOOP map reduce functions we can reduce the computation complexity while implementing the deep learning algorithms. Many deep learning platforms like Caffe support OpenCL and CUDA. We have found that we can use develop applications using these platforms and map them to FPGA rather than use them on GPUs which consume more power. For example, Altera OpenCLSDK support OpenCL development, i.e the OpenCL code can be mapped to FPGA fabric, this reduces the complexity involved in design of RTL code. For accelerating these algorithms, we can choose platforms based on our requirements like computation power required power consumption, time for development etc and improve the performance by using/combing multiple techniques we have that have been discussed in this paper.

# References

1. Intel C++ Intrinsics Reference, http://cache-www.intel.com/cd/00/00/34/76/347603 347603.pdf
2. Eigen, a C++ template library for linear algebra, http://eigen.tuxfamily.org/
3. Jurgen Fritsch and Ivica Rogina (1996) The bucket box intersection (BBI) algorithm for fast ̈approximative evaluation of diagonal mixture Gaussians, Proceedings of ICASSP’96.
4. Kate M. Knill, Mark J.F. Gales, and Steve J. Young (1996) Use of Gaussian selection in largevocabulary continuous speech recognition using HMMs, Proceedings of ICSLP’96.
5. Henggang Cui, Hao Zhang, Gregory R. Ganger, Phillip B. Gibbons and Eric P. Xing. GeePS: Scalable deep learning on distributed GPUs with a GPU-specialized parameter server. In [EuroSys, 2016](http://eurosys16.doc.ic.ac.uk/).
6. FPGA-Accelerated Hadoop Cluster for Deep Learning Computations” Abdulrahman Alhamali, Nibal Salha, Raghid Morcel, Mazen Ezzeddine, Omar Hamdan, Haitham Akkary, and Hazem Hajj, 2015 IEEE 15th International Conference on Data Mining Workshops
7. Collobert, Ronan, et al. "Natural language processing (almost) from scratch."The Journal of Machine Learning Research 12 (2011): 2493-2537.
8. Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).
9. Le, Quoc V. "Building high-level features using large scale unsupervised learning." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.
10. Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
11. Hadsell, Raia, et al. "Learning long\_range vision for autonomous off\_road driving." Journal of Field Robotics 26.2 (2009): 120-144.
12. Holi, Jordan L., and Jenq-Neng Hwang. "Finite precision error analysis of neural network hardware implementations." Computers, IEEE Transactions on 42.3 (1993): 281-290.
13. Chen, Tianshi, et al. "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning." ACM SIGPLAN Notices.
14. J. E. Stone, D. Gohara, and G. Shi, “Opencl: A parallel programming standard for heterogeneous computing systems,” Computing in science & engineering, vol. 12, no. 1-3, pp. 66–73, 2010.
15. J. Lu, S. Young, I. Arel, and J. Holleman, “A 1 TOPS/W analog deep machine-learning engine with ﬂoating-gate storage in 0.13 m CMOS,” IEEE J. Solid-State Circuits, vol. 50, no. 1, pp. 270–281, Jan. 2015.
16. F.Bonomietal.,“Fogcomputinganditsroleintheinternetofthings,” in Proc. 1st Edition MCC Workshop Mobile Cloud Computing, 2012, pp. 13–16.
17. C. T. Chu, S. K. Kim, and Lin, “Map-reduce for machine learning onmulticore,” in Proceedings of the 20th Annual Conference on NeuralInformation Processing Systems (NIPS), 2006, pp. 281–288.
18. R. Raina, A. Madhavan, and A. Y. Ng, “Large-scale deep unsupervisedlearning using graphics processors,” in Proceedings of the 26th Inter-national Conference on Machine Learning (ICML), 2009, pp. 873–880.
19. Y. You, S. L. Song, H. Fu, A. Marquez, M. M. Dehnavi, K. Barker,K. W. Cameron, A. P. Randles, and G. Yang, “Mic-svm: Designing ahighly efficient support vector machine for advanced modern multi-coreand many-core architectures,” in Proceedings of the 28th InternationalParallel and Distributed Processing Symposium (IPDPS), 2014, pp. 809– 818.
20. “Self-driving car test: Steve mahan,” 2012, [Accessed: 2015-04-15].
21. J. Rendeirs, An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors, 2015, [Accessed:2015-03-20].[Online].Available: <https://software.intel.com/sites/default/files/article/> 330164/an-overview-of-programming-for-intel-xeon-processors-and-intel-xeon-phi-coprocessors 1.pdf
22. B. Recht, C. Re, S. J. Wright, and F. Niu, “Hogwild: A lock-free approach to parallelizing stochastic gradient descent.” in NIPS, 2011,pp. 693–701
23. M. Ranzato, C. Poultney, S. Chopra, and Y. LeCun, "Efficient learning of sparse representations with an energy-based model," inAdvances in Neural Information Processing Systems 19 (NIPS'06),(B. Schölkopf, J. Platt, and T. Hoffman, eds.), pp. 1137-1144, MIT Press, 2007
24. Ackley, David H., Geoffrey E. Hinton, and Terrence J. Sejnowski."A learning algorithm for Boltzmann machines." Cognitive science 9.1 (1985): 147-169.
25. T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project Adam: Building an efficient and scalable deep learning training system. In OSDI, 2014.
26. Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In Computer Vision and Pattern Recognition. CVPR ‘09.
27. LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. 1998. Gradient-based learning applied to document recognition. In Proceedings of the IEEE, 86(11):2278–2324, (Nov. 1998).

28.”FPGA-Accelerated Hadoop Cluster for Deep Learning Computations”Abdulrahman Alhamali, Nibal Salha, Raghid Morcel, Mazen Ezzeddine, Omar Hamdan, Haitham Akkary, and Hazem Hajj, 2015 IEEE 15th International Conference on Data Mining Workshops

29. Collobert, Ronan, et al. "Natural language processing (almost) from scratch."The Journal of Machine Learning Research 12 (2011): 2493- 2537.