Aspects of programming for implementation of convolutional neural networks on multisystem HPC architectures

Sunil Pandey¹, Naresh Kumar Nagwani², Shrish Verma³

¹Research Scholar, Dept of Computer Science and Engineering, NIT Raipur, India
²Associate Professor, Dept of Computer Science and Engineering, NIT Raipur, India
³Professor, Dept of Electronics and Communications Engineering, NIT Raipur, India
sys_admin@nitrr.ac.in

Abstract. The training of deep learning convolutional neural networks is extremely compute intensive and takes long times for completion, on all except small datasets. This is a major limitation inhibiting the widespread adoption of convolutional neural networks in real world applications despite their better image classification performance in comparison with other techniques. Multidirectional research and development efforts are therefore being pursued with the objective of boosting the computational performance of convolutional neural networks. Development of parallel and scalable deep learning convolutional neural network implementations for multisystem high performance computing architectures is important in this background. Prior analysis based on computational experiments indicates that a combination of pipeline and task parallelism results in significant convolutional neural network performance gains of up to 18 times. This paper discusses the aspects which are important from the perspective of implementation of parallel and scalable convolutional neural networks on central processing unit based multisystem high performance computing architectures including computational pipelines, convolutional neural networks, convolutional neural network pipelines, multisystem high performance computing architectures and parallel programming models.

1. Introduction
Convolutional neural networks are an important deep learning paradigm which have demonstrated superior performance and have typically been used for applications involving artificial visual intelligence such as in computer vision, object detection, pattern recognition, image classification, scene understanding, etc. in different applied engineering disciplines. Convolutional neural networks have seen a range of applications in different domains [1-20] over the last decade and have demonstrated superior accuracies in classification problems when compared to other algorithms on the same task [21-24]. For a general introduction to deep learning and convolutional neural networks, the reader is referred to [25]. However, there is a general view among the researchers and practitioners of the artificial intelligence community that the adoption of the convolutional neural network for real-world applications is being hindered, in large part, due to their long training times which sometimes range from weeks to even months on advanced computer architectures [26-28]. Accordingly, research and development activities are being taken up in multiple directions to redress this problem [29-44].
The design and development of parallel and scalable deep learning algorithms for central processing unit based multisystem high performance computing architectures is an important direction being pursued in this context. This paper discusses the aspects which are crucial from the perspective of implementation of the convolutional type deep learning networks on central processing unit based multisystem high performance computing architectures.

The paper begins with a brief introduction to computational pipelines and pipeline parallelism. The computational primitives and architecture of convolutional neural networks are taken up next for discussion followed by an introduction to the convolutional neural network pipeline. This is followed by a discussion of the multisystem parallel machine model followed by an introduction to the parallel programming model useful for parallel implementation of convolutional neural network pipelines on multisystem parallel machines.

2. Computational Pipelines

In a pipeline like computation design, a relatively large computation is broken down into two or more smaller computations and these smaller computations are arranged in a sequence, one after the other, with the output of the previous block becoming the input to the current block, and the output of the current block becoming the input to the next block, and so on so forth. Figure 1 shows a computational pipeline. This is true for all but the source and sink blocks which represent the source inputs and the final outputs respectively. Different aspects of computational pipelines are discussed in [45-50].

![Figure 1. Computational Pipeline with Five Tasks](image)

3. Pipeline Parallelism

The pipeline implementation of an algorithm makes it appropriate for the time tested parallel computing paradigm of pipeline parallelism. Pipeline parallelism can itself be viewed as a kind of functional or task decomposition and segmented computing. In pipeline parallelism, the computational problem is divided into separate functions or tasks which can be executed individually and independently. These computational functions or tasks can also be mapped to independent cores of a central processing unit based multisystem high performance computing architecture.

Figure 2 shows the space-time diagram of a relatively simple computational pipeline of Type 1. As can be observed from the figure, the different pipeline stages require equal times for execution. The speedup on a computation so divided can be calculated for the simple case of equal stage pipeline and is given in equation 1 where \( p \) is the number of pipeline stages and \( m \) is the number of data samples.

\[
\text{Speedup} = \frac{mp}{m + (p-1)}
\]
Figure 2. Space-Time Diagram of a Simple Computational Pipeline

4. Computational Primitives and Architecture of CNN

Convolutional neural networks are natural candidates for pipelining due to their stacked computational layer architecture.

![Figure 3. Architecture of a typical convolutional neural network](image)

There are four important computational layers or computational primitives associated with a typical convolutional neural network as can be seen from Figure 3 that depicts a typical convolutional neural network architecture with the dog Sundari’s image as the input. They include the convolution, nonlinear activation, pooling or subsampling and the fully connected layers. The first three layers are used for feature extraction from the input images while the last one is used in classification. The convolutional layer is used for raw feature extraction through the convolution operation, the nonlinear activation or mapping layer is used for nonlinear transformation and mapping of the inputs to a normalized form, the pooling layer is mainly used for dimensional reduction. The fully connected layer is the final component of the convolutional neural network which is used for supervised learning of the mapping between the inputs and the corresponding desired outputs and training of the classifier stage.

In the convolutional layer, a two dimensional convolution is performed between the image matrix corresponding to one of the channels and a kernel which is essentially a small odd dimensioned square matrix. Zero padding is applied to maintain the dimensions of the original image channel matrix. In the activation layer, the inputs are passed through a nonlinear function which is typically one of the ReLU, hyperbolic tangent, absolute hyperbolic tangent, or the sigmoid function. In pooling or subsampling, the image is divided into non-overlapping regions of say, 2 x 2. The maximum or the average of the numbers in a specific region replaces the region. This results in significant dimension reduction of the feature space. With 2 x 2 pooling, for example, the image is reduced to 25% of its original size. In the fully connected layer, the flattened image features at the end of the feature
generation step serve as the input to a supervised feedforward error backpropagation network which maps the image to the desired output class during the training step. The network is trained using the gradient descent optimization algorithm. [45]. Figs 4(a) – 4(d) demonstrate the four different computational primitives of the convolutional neural network.

![Figure 4](image)

**Figure 4.** (a) Convolution Layer; (b) ReLU Activation Layer; (c) Max Pooling Layer; (d) FC Layer

5. **Convolutional Neural Network Pipeline**

Figure 5 shows a well-known convolutional neural network, the LeNet (left) and highlights its pipeline architecture (right).

![Figure 5](image)

**Figure 5.** LeNet (left) and Equivalent Convolutional Neural Network Pipeline (Right)

The pipeline flow in the case of LeNet can be described in the following manner. Input to the LeNet is a 28 pixels x 28 pixels image. The first stage is the convoluter stage which comprises of 6
convolutional filters of size 5 x 5 and a padding of 2. The second stage is the pooler or sub-sampler stage comprising of non-overlapping average pooling windows of size 2 x 2 and a stride of 2. The third stage is a convoluter stage comprising of 16 convolutional filters of size 5 x 5 without padding. The fourth stage and the second stages are similar. The fifth, sixth and seventh stages are the input, hidden and output layers of a three-layer artificial neural network with 120, 84, and 10 artificial neurons in each of the layers, respectively.

Figure 6. Sample Convolutional Neural Network Feature Extraction Pipeline

Figure 6 shows a sample convolutional neural network feature extraction pipeline. The pipeline can be seen to be comprised of one loader block which is used for loading the instances of images from the image dataset. It also comprises of four sets of convolution convoluter1 through convoluter4, nonlinear mapping, nonlinear1 through nonlinear4 and pooling, pooling 1 through pooling4 blocks. The four convolution blocks also make use of four sets of filter blocks filt1 through filt4 blocks. The flattening block at the end is used to flatten 2-D matrices into 1-D feature vectors.

An analysis of the design of convolutional neural networks as computational pipelines and their implementation using the task parallel model on central processing unit based multisystem high performance computing architecture reveals that notable performance gains in the form of speedup factors of up to 18X are achievable [51]. A multicore implementation of the convolution operation can further improve the computational performance [52]. Performance gains are influenced by factors like type of problem, type of dataset, type of high performance computing architecture, type of mapping and the pipeline design.
6. Multisystem Parallel Machine Model

The multisystem parallel machine model can be treated as a parallel extension of the classic Von Neumann sequential computer model. The Von Neumann computer is an abstract model which abstracts or captures the essentials of a sequential computing machine. It has a central processing unit connected to a “storage”, which is the memory. The central processing unit executes a “stored program” which is a sequence of reads and writes on the connected memory.

![Multisystem Parallel Computer Model](image)

**Figure 7. Multisystem Parallel Computer Model**

The multisystem or the multicomputer model [53] can be considered as a parallel extension of the sequential Von Neumann model. As can be seen from Figure 7(b), the multisystem model consists of two or more Von Neumann computers, which can be called nodes, which are linked by an interconnection network. A program written for the multisystem can access the local memory and can also communicate with other nodes, i.e., read and write to remote memories of other nodes by sending and receiving messages over the network. If the network is idealized then an assumption can be made that the cost of sending a message between two nodes is independent of the location or other network traffic but depends on the length of the message only.

It is a characteristic of the parallel multisystem model that “local reads and writes” are less expensive than “remote sends and receives”. This property called locality is an important consideration when writing parallel programs besides the requirements of concurrency and scalability. For the reason of locality it is a desirable characteristic in parallel programs that the local data accesses are more frequent than the remote data accesses. The multisystem or multicomputer model comes closest in practice to the distributed memory multiple instruction multiple data. In a multiple instruction multiple data parallel machine, each processor can execute an independent stream of instructions on its local data. Distributed memory means that the memory is distributed across the central processing units rather than being centralized. In the real world, the central processing unit based multisystem high performance computing architecture generally takes the form of a server cluster in which a number of servers called cluster nodes communicate with each other through a cluster interconnect. The interconnect may be dedicated to the cluster in which case the cluster is considered tightly coupled or general purpose in which case it is considered loosely coupled.

7. Parallel Programming Model

The parallel programming model described here is based on the concept of tasks and channels [53] for programming central processing unit based multisystem or multi-computer architectures. The basics of the tasks and channels model is described in the following points.

1. A parallel computation is comprised of one or more tasks which have the capability to execute concurrently.
2. Tasks can vary in number when a program is executing.
3. A task refers in totality to a sequential program, local memory, and a set of inports and outports which define its interfaces.
4. A task can perform the following four basic actions besides reads and writes to its local
memory: send messages on its outports, receive messages on its inports, create new tasks, and terminate.

5. The send operation is an asynchronous operation in that it completes immediately. However, a receive operation is synchronous. It causes the task execution to block until a message becomes available.

6. The outport and inport pairs can be connected by message queues called channels. Channels can be created and deleted, and references to channels with port information are part of the messages, therefore the connectivity is dynamic during the program execution.

7. It is possible to map tasks to physical processors in different ways. However, choice of a particular mapping does not have any effect on the program semantics. It is possible for multiple tasks to be mapped to a single physical processor.

Figure 8 illustrates the task and channel parallel programming model pictorially.

![Figure 8. Task and Channel Parallel Computation Model](image)

Figure 9 illustrates four basic actions that are performed by a task in the task and channel parallel model.

![Figure 9. The Four Basic Actions of a Task in the Task and Channel Model](image)

8. Conclusions
The paper discusses aspects of computational pipelines, convolutional neural networks, implementation of the convolutional neural network as a computational pipeline, central processing unit based multisystem high performance computing architectures and parallel programming models which are important from the perspective of implementing deep learning convolutional type neural networks on central processing unit based multisystem high performance computing architectures. The paper begins with an explanation of computational pipelines and pipeline parallelism. The computational primitives of the convolutional neural networks are taken up next for discussion followed by an introduction to the convolutional neural network pipelines. Subsequently, there is a discussion of the multisystem parallel machine model followed by an introduction to the parallel programming model which is used for implementation. The paper therefore touches on all aspects
critical from the perspective of implementation of parallel and scalable convolutional type deep learning networks on the central processing unit based multisystem high performance computing architectures.

References

[1] Zezhou Cheng Qingxiong Yang and Bin Sheng Deep Colorization IEEE International Conference on Computer Vision (ICCV) Santiago 2015 pp. 415-423.
[2] Zhang R. Isola P. and Efros A.A. Colorful Image Colorization European Conference on Computer Vision 2016 - Springer vol. 9907 2016.
[3] Larsson G. Maire M. and Shakhnarovich G. Learning Representations for Automatic Colorization Computer Vision( ECCV) Lecture Notes in Computer Science - Springer vol. 9908 2016.
[4] Hwang Jeff and You Zhou Image Colorization with Deep Convolutional Neural Networks Stanford University 2016.
[5] Andrew Owens et al. Visually Indicated Sounds in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Las Vegas 2016 pp. 2405 - 2413.
[6] I Sutskever O. Vinyals and Q.V. Le Sequence to Sequence Learning in Proc. Advances in Neural Information Processing Systems 27 2014 pp. 3104–3112.
[7] K. Cho et. al. Learning phrase representations using RNN encoder-decoder for statistical machine translation in Proc. Conference on Empirical Methods in Natural Language Processing 2014 pp. 1724–1734.
[8] Zhang Jiajun and Zong Chengqing Deep Neural Networks in Machine Translation: An Overview IEEE Intelligent Systems vol. 30 no. 5 pp. 16-25 Sept-Oct 2015.
[9] Krizhevsky I. Sutskever and G. Hinton ImageNet classification with deep convolutional neural networks in NIPS Proceedings 2012.
[10] G. Howard Some improvements on deep convolutional neural network based image classification in International Conference on Learning Representation (ICLR) Banff Canada 2014.
[11] D. Erhan C. Szegedy A. Toshev and D. Anguelov Scalable Object Detection Using Deep Neural Networks in IEEE Conference on Computer Vision and Pattern Recognition Columbus 2014 pp. 2155-2162.
[12] D. Erhan C. Szegedy and A. Toshev Scalable object detection using deep neural networks in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2014.
[13] Alex Graves Generating Sequences With Recurrent Neural Networks 2014. [Online]. https://arxiv.org/pdf/1308.0850.pdf
[14] Ilya Sutskever James Martens and Geoffrey E Hinton Generating text with recurrent neural networks in Proceedings of the 28th International Conference on Machine Learning (ICML-11) New York NY 2011 pp. 1017-1024.
[15] Andrej Karpathy and Li Fei-Fei Deep Visual-Semantic Alignments for Generating Image Descriptions in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Boston Massachusetts 2015 pp. 3128 - 3137.
[16] Ayushi Chahal Preeti Gulia Deep Learning: A Predictive IoT Data Analytics Method International Journal of Engineering Trends and Technology vol. 68 no. 7 2020.
[17] P. Seetha Subha Priya S. Nandhinidevi M. Thangamani S. Nallusamy A Review on Exploring the Deep Learning Concepts and Applications for Medical Diagnosis International Journal of Engineering Trends and Technology vol. 68 no. 10 2020.

[18] Sangeeta Preeti Gulia Deep learning based combating strategy for COVID-19 induced increased video consumption International Journal of Engineering Trends and Technology vol. 68 no. 7 2020.

[19] Ferdinand Kartriku Robert Sowah Charles Saah Deep Neural Network: An Efficient and Optimized Machine Learning Paradigm for Reducing Genome Sequencing Error International Journal of Engineering Trends and Technology vol. 68 no. 9 2020.

[20] Ramya T.E. Marikkannan M. Investigations on Combinational Approach for Processing Remote Sensing Images Using Deep Learning Techniques International Journal of Engineering Trends and Technology vol. 67 no. 8 2019.

[21] Ciresan Dan Ueli Meier and Jürgen Schmidhuber. 2012. Multi-column deep neural networks for image classification. In IEEE Conference on Computer Vision and Pattern Recognition 2012 pp. 3642-3649.

[22] Ciresan Dan Ueli Meier Jonathan Masci Luca M. Gambardella and Jurgen Schmidhuber. 2011. Flexible High Performance Convolutional Neural Networks for Image Classification. in International Joint Conference on Artificial Intelligence 2013 pp. 1237–1242.

[23] Lawrence Steve C. Lee Giles Ah Chung Tsoi and Andrew D. Back.Face Recognition: A Convolutional Neural Network Approach. IEEE Transactions on Neural Networks 1997 Volume 8; Issue 1 pp. 98-113

[24] Russakovsky O. Deng J. Su H. et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis vol. 115 2015

[25] Yann LeCun Yoshua Bengio and Geoffrey Hinton Deep Learning Nature vol. 521 p. 436 2015.

[26] X. W. Chen and X. Lin Big Data Deep Learning: Challenges and Perspectives IEEE Access vol. 2 pp. 514-525 2014.

[27] M.M. Najafabadi et. al. Deep learning applications and challenges in big data analytics Journal of Big Data p. 1 2015.

[28] P. Angelov and A. Sperduti Challenges in Deep Learning in European Symposium on Artificial Neural Networks Computational Intelligence and Machine Learning Bruges (Belgium) 2016 pp. 27-29.

[29] Xian-He Sun Yong Chen and Surendra Byna Scalable Computing in the Multicore Era in Proceedings of the Inaugural Symposium on Parallel Algorithms Architectures and Programming Hefei: University of Science and Technology of China Press 2008.

[30] Sunil Pandey Naresh Kumar Nagwani Shrish Verma. Comparative Evaluation of Performance and Scalability of Convolutional Neural Network Implementations on a Multisystem HPC Architecture. Accepted in International Conference on Computational Intelligence and Energy Advancements ICCIEA-2021

[31] M. B. Giles and I. Reguly Trends in high-performance computing for engineering calculations in Philosophical Transactions of the Royal Society of America 2014.

[32] J. Dean et. al. Large Scale Distributed Deep Networks in 25th International Conference on Neural Information Processing Systems Lake Tahoe Nevada 2012 pp. 1223-1231.
[33] J. Hauswald et. al. DjiNN and Tonic: DNN as a service and its implications for future warehouse scale computers in Proceedings 42nd Annual International Symposium on Computer Architecture (ISCA) Portland OR USA 2015 pp. 27-40.

[34] V. Hegde and S. Usmani. (2016) Parallel and Distributed Deep Learning Stanford University Online Report.

[35] M. Bouache and J. Glover Deep Learning GPU-Based Hardware Platform Hardware and Software Criteria and Selection in ICS-2016 Istanbul Turkey 2016.

[36] Q. Le et. al. On Optimization Methods for Deep Learning in Proceedings of the International Conference on Machine Learning Washington 2011.

[37] J. Keuper and F.J. Pfreibndt Asynchronous Parallel Stochastic Gradient Descent A Numeric Core for Scalable Distributed Machine Learning Algorithms in Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments Austin TX USA 2015.

[38] V. Vanhoucke A. Senior and M. Z. Mao Improving the speed of neural networks on CPUs in Proceedings of the Deep Learning and Unsupervised Feature Learning NIPS Workshop Granada Spain 2011.

[39] S. Gupta A. Agrawal K. Gopalakrishnan and P. Narayanan Deep Learning with Limited Numerical Precision Journal of Machine Learning Research 37 2015.

[40] S Chetlur et. al. (2014) cuDNN: Efficient Primitives for Deep Learning. [Online]. https://arxiv.org/pdf/1410.0759.pdf

[41] Delong. Practical Guide to Matrix Calculus for Deep Learninghttp://www.psi.toronto.edu/~andrew/papers/matrix_calculus_for_learning.pdf

[42] Baoyuan Liu Min Wang H. Foroosh M. Tappen and M. Penksy Sparse Convolutional Neural Networks in IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Boston MA 2015 pp. 806-814.

[43] Ionescu O. Vantzos and C. Sminchisescu Matrix Backpropagation for Deep Networks with Structured Layers in IEEE International Conference on Computer Vision (ICCV) Santiago 2015 pp. 2965-2973.

[44] Y. Zhang and S. Zhang Optimized Deep Learning Architectures with Fast Matrix Operation Kernels on Parallel Platform in 2013 IEEE 25th International Conference on Tools with Artificial Intelligence Herndon VA 2013 pp. 71-78.

[45] Mario Rossainz-López Manuel I. Capel Odon D. Carrasco-Limón Fernando Hernández-Polo Bárbara E. Sánchez-Rinza Implementation of the Pipeline Parallel Programming Technique as an HLPC: Usage Usefulness and Performance Annals of Multicore and GPU Programming 4 (1). ISSN: 2341-3158.

[46] I-Ting Angelina Lee Charles E. Leiserson Tao B. Schardl Jim Sukha Zhunping Zhang On-the-Fly Pipeline Parallelism ACM Transactions on Parallel Computing 2 No. 3

[47] Halling-Brown M Shepherd AJ. Constructing computational pipelines. Methods Mol Biol. 2008;453:451-70.

[48] Vivien Marx When Computational Pipelines Go Clank. Nature Methods 17 659–662 (2020)

[49] Saurav Chatterjee and Jay Strosnider Distributed Pipeline Scheduling: A Framework for Distributed Heterogeneous Real-Time System Design The Computer Journal 38 No. 4 1995
[50] Navarro R. Asenjo S. Tabik and C. Cascaval Analytical Modeling of Pipeline Parallelism 2009 18th International Conference on Parallel Architectures and Compilation Techniques 2009 pp. 281-290 doi: 10.1109/PACT.2009.28.

[51] Sunil Pandey Naresh Kumar Nagwani Shrish Verma Analysis and Design of High Performance Deep Learning Algorithm: Convolutional Neural Networks International Journal of Engineering Trends and Technology 69.6(2021):216-224.

[52] Sunil Pandey Naresh Kumar Nagwani Shrish Verma. Parallel and Scalable Deep Learning Algorithms for High Performance Computing Architectures International Journal of Engineering Trends and Technology 69(4) 236-246. (2021)

[53] Ismail Fauzi Isnin High Performance and Parallel Computing Chapter 3 on Parallel Algorithm Design https://people.utm.my/nur-haliza/files/2020/09/04-parallel-algorithm-design-published.pdf

[54] M. Tanveer M.A. Iqbal and F. Azam Using Symmetric Multiprocessor Architectures for High Performance Computing Environments International Journal of Computer Applications vol. 27 no. 9 2011.