ISSN XXXX XXXX © 2016 IJESC

Research Article Volume 6 Issue No. 12

GPU Accelerated Image Segmentation using Cascaded Hierarchical Models

K Chaitanya Pavan Tanay[1], K Yogesh Murthy[2], P D Kartheek[3], Pallav Kumar Baruah[4]
M.Tech Student[1,2,3], Associate Professor[4]
Department of Mathematics and Computer Science
Sri Sathya Sai Institute of Higher Learning, Prasanthi Nilayam, India

Abstract:
Image segmentation is a fundamental problem in image processing that aids in the analysis and interpretation of images. All pixels in the image that are similar with respect to some property are grouped together into a single segment. The discovery of similar pixels can be enhanced by incorporating contextual information into the segmentation process. Further, using information about the context makes segmentation robust to noise. However, two major issues arise during such an attempt. Firstly, unsupervised extraction of contextual information from an image is a difficult problem, and secondly, there are no formal techniques that describe how to effectively utilize the extracted contextual information. Seyedhosseini, et al. [1] present a hierarchical framework for learning contextual information in an image. In this method, each level consists of a classifier that is trained on the outputs of the previous level and a down-sampled version of the input image. The contextual information obtained from this classifier is then used for segmenting the input image at its original resolution. The classifier used in this method consists of a layer of logistic sigmoid functions that perform adaptive feature detection and a pair of fixed layers that are composed of logical units to perform conjunctions and disjunctions. This procedure is evidently sequential, with the parameters of the classifier at each level dependent on the outputs from the preceding levels. In this work, we parallelize the computation of discriminants for the classifiers on GPUs, achieving a speedup of 400X without any loss in accuracy during testing, as compared to the original CPU-only work.

Keywords: GPU, Image Segmentation, unsupervised, logistic sigmoid.

I. INTRODUCTION

Image segmentation is typically the first step in image understanding and interpretation in which an input image is transformed into a set of segments, each consisting of pixels with similar attributes. Thesuccess of a high-level image analysis task dependson an accurate segmentation, and hence, the quality of the segmentation algorithm becomes a crucial factor. Moreover, being used most often as a pre-processing step, an image segmentation algorithm cannot take too much time for computation. Segmentation techniques can be broadly classified into two: *contextual* and *non-contextual*. The non-contextual techniques group pixels based on the value of a global attribute such as pixel intensity value, and do not consider the spatial relationships among pixels. Contextual information could refer to either inter-object or intra-object relationships. For example, the presence of a car bonnet in an image could indicate the presence of its wheels (inter-object), as well as the likely presence of other cars (intra-object). In this work, we modify the segmentation framework presented in [1] that exploits features in the input image along with additional contextual information. The model consists of a hierarchy of classifiers trained on the outputs of classifiers in preceding levels and a down-sampled version of the input image. Since there are lots of classifiers to be trained, the large number of free parameters renders the training step slow and susceptible to overfitting due to noise. To overcome this, a simplified probabilistic classifier, called the Logistic Disjunctive Normal Networks (LDNN), is used. Contrary to traditional neural networks, this classifier has a single adaptive layer consisting of logistic sigmoid functions. Additionally, there are two logical unit layers that perform disjunctions and conjunctions, respectively. The classifier updates its weights at each step using the gradient descent method.Though it is easier and faster to train the LDNN than traditional neural nets, the whole approach is still sequential. In this work, we reach the maximum achievable speedup from this model by exploiting fine-grained parallelism inherent in the gradient descent approach. More specifically, we incorporate elements of the stochastic gradient descent method [2] into the approach of [1] to achieve speedups of upto 400x over the original work.

II. RELATED WORK

Many techniques have been proposed to augment contextual information with image segmentation and scene understanding. Choi *et al.* [4] propose a probabilistic graphical tree structured architecture to model object co-occurrences and spatial relationships in order to capture the inherent structure among different object categories. Combining the output of this model with local detectors and image features, all instances of multiple object categories are detected and localized. He *et al.* [5] propose the use of Conditional Random Fields to create a different spatial scales label which are obtained from labeled image set. These encode both global image-level contextual information as well as finer local patterns. Torralba *et al.* [6] use boosting approach to learn the graphical structure of Conditional Random Fields by pooling information from large regions of the image in an additive fashion, creating a data structure called Boosted Random Fields (BRF). The idea of organizing black-box classifiers for scene categorization, multi-class image segmentation and object detection synergistically into tiers, where the output of one tier is fed into successive tiers, was proposed by Heitz *et al.* [7] as the Cascaded Classification Model. Desai *et al.* [8]

introduced a discriminative model for multi-class object recognition, leading to simultaneous prediction of labels for all categories of objects in an image. Tu[9] propose a discriminative algorithm that learns a cascade of classfiers for segmentation. Stochastic Gradient Descent (SGD) is a popular algorithm for machine learning, deep learning and big data problems that handle large size data inputs since it converges quickly, doesn't have high memory requirements and is robust to noise in the data. Since MapReduce-like frameworks have become the de facto standard for pre-processing big data problems, most of the work on parallelizing SGD is in that direction [3]. However, the inherent sequential nature of SGD makes synchronization (locking) the bottleneck for any parallel SGD algorithm. In [2], Niu *et al*. propose a lock-free parallel implementation of SGD with the assumption of sparsity of accesses to the shared data.

III. OVERVIEW OF OUR WORK

Flowchart of Algorithm

The two major parts of the model presented in [1] are the Cascaded Hierarchical Model (CHM) and the Logistic Disjunctive Normal Networks (LDNN).

Cascaded Hierarchical Model:

The Cascaded Hierarchical Model (CHM) consists of repeating the following steps consecutively.

- The input image is sequentially down-sampled to obtain a multi-resolution representation.
- At each resolution from the finest to the coarsest, a classifier is trained based on the output of the classifier at the previous level of the hierarchy and the input image at that resolution.
- A new classifier is trained for the image at its original resolution based on the outputs of the classifiers at various levels of the hierarchy.

Given a 2D input image for segmentation, the following operations are performed during the steps of CHM.

- $\Phi(.,h)$:Perform down-sampling h times by averaging in 3 × 3 window.
- $\Psi(.)$:Extract features.
- $\tau(.,h)$:Perform max-pooling operation h times to find maximum pixel value in each 3 × 3 window.
- $\Omega(.,h)$:Perform up-sampling h times by duplicating each pixel.

The internal parameters θ_h are learned during training by each classifier in the hierarchy. The internal parameters of final classifier β are learned by using the outputs of the classifiers at various levels of the hierarchy along with the input image at its original resolution. The output of β denotes the computed label of the input. Since the computation of β uses information from various scales of the input, both local and global information are incorporated into its calculation, making it robust to noise. The CHM model can be summarized by Figure 1 from [1]. The red rectangles denoted in the figure are the classifiers used in the CHM model, that occupy major computation time in the entire algorithm.

Logistic Disjunctive Normal Networks:

Using the property that Boolean functions can be written as a disjunction of conjunctions, the binary classification problem can be approximated as the union of convex sets formed by the intersection of arbitrary half-spaces in R^n, and represented as a disjunction of conjunctions of the half-spaces.

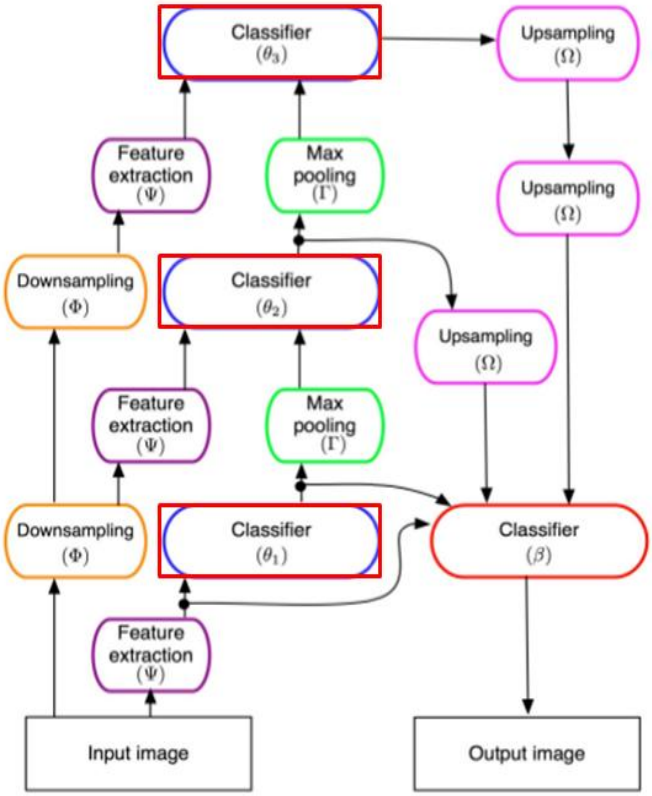


Figure. 1. CHM Model

With further relaxations of the conditions and Boolean manipulations, the half spaces can be approximated by logistic sigmoid functions, giving a differentiable disjunctive normal form approximation to the Boolean function f.

$$f'(X)=1-\prod_i(1-\Pi_j\sigma_{ij}(X)) \quad (1)$$

This can be viewed as a 3-layer network. The first layer consists of N groups with M nodes each that map the input vector X by a sigmoid function σ_{ij}. Each group in the first layer is connected to a single node in the second layer that implements negation. The third layer is a single node that implements disjunction. This is called a $N \times M$ LDNN. Given the training set (X, y), the weights of the LDNN are computed by minimizing the quadratic error over the training set. The method used for the minimization procedure is the gradient descent method. In our work, we replace the gradient descent implementation in [1] by a GPU-based parallel version of the stochastic gradient descent given in [2].

The steps in our algorithm are as follows:

- Randomly shuffle the input data across various thread blocks.
- Initialize weights to 0.
- For each item in the input, apply the gradient descent method to compute the weights.

- Aggregate the weights from all the thread blocks, computing the final weights.

Profiling Analysis

Profiling analysis of the sequential implementation of [1] shows that a major portion of the execution time is spent in constructing classifiers in each iteration of CHM. More specifically, the minimization of the quadratic error over the training set during classification using gradient descent is revealed to be the bottleneck.

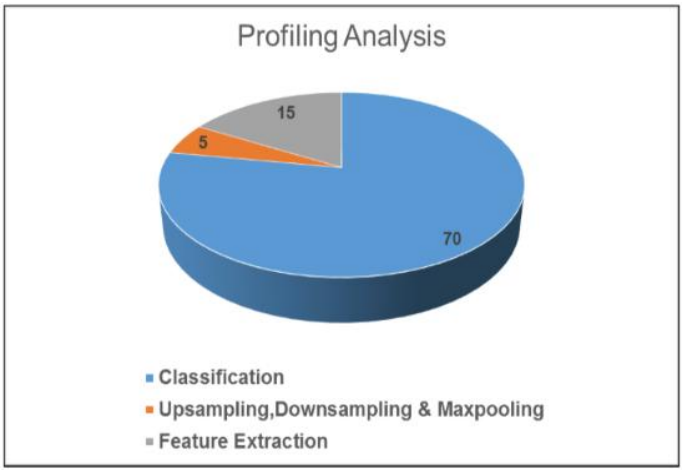


Figure . 2. Profiling Analysis

From Figure 2, we can infer that about 70% of computation time is spent on classification. In order to reduce computation time and gain speedup, we port the sequential CPU-based code for discriminant computation using gradient descent to GPU using ideas from [2], a lock-free approach to parallelizing stochastic gradient descent.

Implementation

We implemented our work in MATLAB MEX since it provides an ideal environment for testing both sequential and parallel programs on the same platform, thus facilitating easy comparison. Further, we used the MATLAB MEX CUDA interface for parallel implementation of CHM on GPU. In this work, we replace the gradient descent implementation in [1] by a GPU-based parallel version of the stochastic gradient descent in [2]. In the flowchart of the algorithm above in Figure 1, the highlighted rectangular boxes with a red border indicate the parallelized steps. The work in [2] assumes a shared memory model in which the decision variable is accessible to all processors that can read and contribute an update vector to it. Latest generations of GPU supports four types of memory global memory, constant memory, shared memory and register memory. Global memory is the slowest and register memory is the fastest in terms of access speed. In the naive GPU implementation each thread reads input data and parameters from the global memory. By preloading the data and its parameters from global to shared memory we improved the speed of the algorithm 2 to 30 times. The use of shared memory model is summarized in Figure 3. The overall speedup achieved using optimized GPU approach is about 400x.

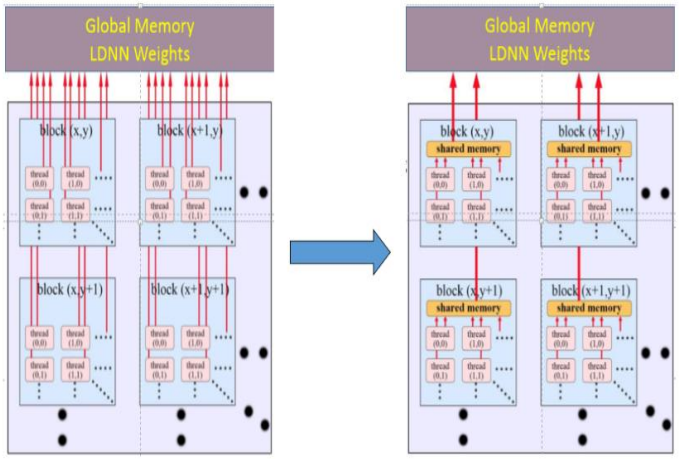


Figure.3. GPU Shared Memory Optimization

The following are the major modules in our implementation.

- Filter bank: Extract features from image
- Learn And Or Net: Find weights of the LDNN
- K Means: Partition the classes in order to obtain initial estimate of weights
- Down Sample: Down-sample the image by averaging in a window
- Max Pooling: Find the maximum pixel value in a window in the image
- Up Sample: Up-sample the image by duplicating pixels
- Update Discriminants: Implemented on GPU, this function finds the values of the weights that minimizes the quadratic error term of the Logistic Disjunctive Normal Networks.

IV. EXPERIMENTAL RESULTS

In this section we present parallel implementation of CHM on GPU's using MATLAB MEX CUDA interface and multicore using OpenMP interface. The hardware used for our experiments is an Intel Core i5-4670 CPU 4 cores@3.40GHz and parallel execution for both multicore and GPU on Intel® Xeon® CPU E5-2609 8 cores@2.40 GHz equipped with NVIDIA GTX Titan X with CUDA 5.2. We used three datasets to compare our performance of optimized GPU over CPU and multicore OpenMP implementations.

- **Mouse Neurophil Dataset**: It consists of 70 images of size 700 by 700 pixels depicting the mouse neuropil. Random selections of 12 images are used for training with the remaining 58 as the test set.
- **Drosophila Dataset:** It contains 30 images of size 512 by 512 pixels from Drosophila first instar larva ventral nerve cord (VNC). We use 14 images for training and 16 images for testing.
- **Weizmann Horse Dataset**: This dataset contains 328 gray scale horse images with foreground and background labels. The problem is to detect membranes in the first two datasets and distinguish the horse from the background in the third.

In Figure 4, we depict a comparison between the execution time of the naïve GPU implementation and shared GPU memory optimized model on the three datasets. We observe that using shared memory gives about 30X speed-up over the global memory model, as expected.

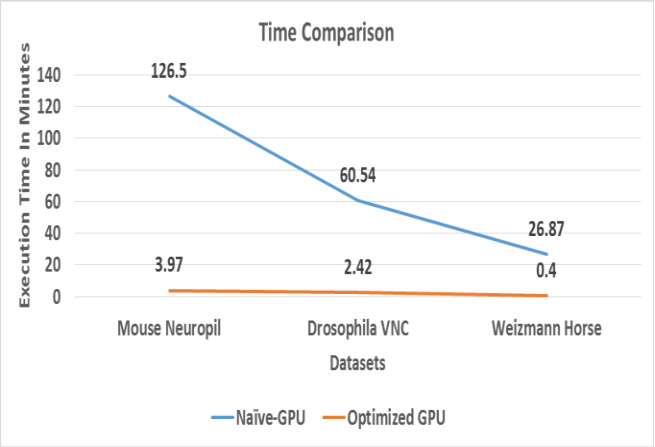


Figure. 4. Time Comparison between Naïve & Optimized GPU

In Figure 5, we compare the execution times of a sequential implementation on CPUs, a shared memory implementation using OpenMP on CPUs and a shared memory implementation using GPUs. We observe average speed-ups of about 110X and 400X using optimized GPUs over the CPU and OpenMP implementations respectively, validating our method of parallelization of the algorithm.

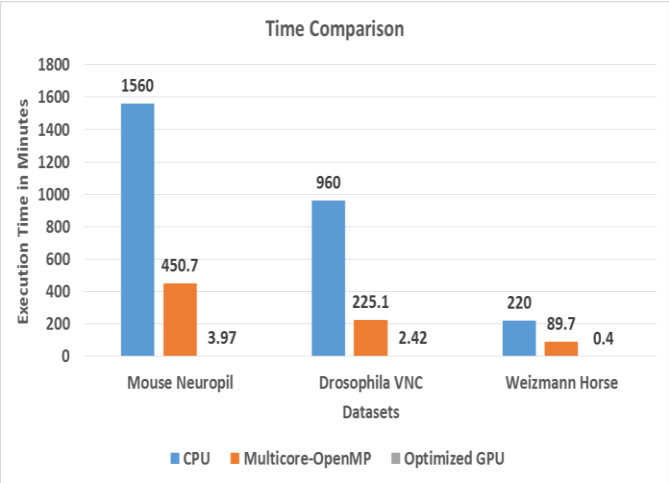
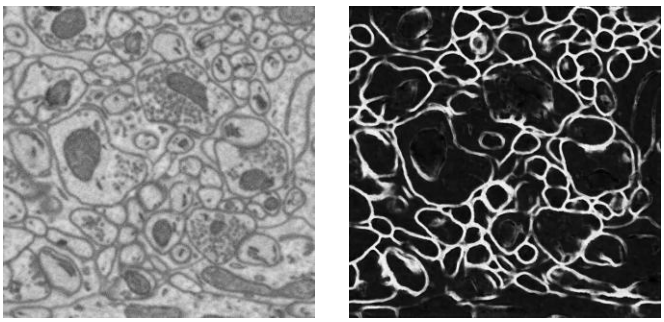


Figure.5. Time Comparison between CPU, OpenMP & Optimized GPU

An example output of our implementation is depicted in Figure 6 below.



Input Image ***Output Image***

Figure. 6. Mouse Neurophil Dataset

The GPU version of the CHM model outperformed the sequential implementation and multi-core OpenMP implementation on all the datasets without impacting the accuracy of results.

V. CONCLUSION

Our implementation of the Cascaded Hierarchical Model for image segmentation using contextual information is more computationally efficient than the sequential work. We further believe that the speedup achieved in our tests is the maximum achievable under the constraints of Amdahl's law. Using a optimized GPU-based implementation, we have achieved a speedup of 400x over the original version. This work can be extended by considering other simple non-linear parametric classifiers which are either computationally more efficient than logistic sigmoid functions or are robust to noise. Further, we can investigate the use of contextual information in simple linear classifiers since a majority of the real world problems can be cast as linearly separable through appropriate transformations. The purpose of this work is not restricted to achieving speedup for a single image processing problem. Rather, it is meant as a demonstration of the methodology by which an entire class of computationally similar problems can be tackled.

VI. ACKNOWLEDGEMENT

Our work is dedicated to Bhagawan Sri Sathya Sai Baba, Founder chancellor of Sri Sathya Sai Institute of Higher Learning, for His inspiration and guidance, and providing the necessary infrastructure. This work is partially supported by the grant under NVIDIA GPU Research Center program.

VII. REFERENCES

[1]. Mojtaba Seyedhosseini, Mehdi Sajjadi, and Tolga Tasdizen,"Image segmentation with cascaded hierarchical models and logistic disjunctive normal networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2168 2175.

[2] .Recht, Benjamin, et al. "Hogwild: A lock-free approach to parallelizing stochastic gradient descent." *Advances in Neural Information Processing Systems*. 2011.

[3]. Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola, "Parallelized stochastic gradient descent," in *Advances in neural information processing systems*, 2010, pp. 2595–2603.

[4]. Choi, Myung Jin, Antonio Torralba, and Alan S. Willsky. "A tree-based context model for object recognition." *IEEE transactions on pattern analysis and machine intelligence* 34.2 (2012): 240-252.

[5]. He, Xuming, Richard S. Zemel, and Miguel Á. Carreira-Perpiñán. "Multiscale conditional random fields for image labeling." *Computer vision and pattern recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE computer society conference on*. Vol. 2. IEEE, 2004.

[6]. Torralba, Antonio, Kevin P. Murphy, and William T. Freeman. "Contextual models for object detection using boosted random fields." *Advances in neural information processing systems*. 2004.

[7]. Heitz, Geremy, et al. "Cascaded classification models: Combining models for holistic scene understanding." *Advances in Neural Information Processing Systems*. 2009.

[8]. Desai, Chaitanya, Deva Ramanan, and Charless C. Fowlkes. "Discriminative models for multi-class object layout." *International journal of computer vision* 95.1 (2011): 1-12.

[9]. Tu, Zhuowen, and Xiang Bai. "Auto-context and its application to high-level vision tasks and 3d brain image segmentation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.10 (2010): 1744-1757.