GPU-Framework for Action Recognition

ABSTRACT

Real time processing for teamwork activity recognition is a challenge due to complex computational models built for achieving high system accuracy. Hence, this paper proposes a framework based on Graphical processing Units (GPUs) to achieve speed–up in the performance of role based activity recognition of teamwork. The framework can be applied in various fields, especially athletic and military applications. Furthermore, the framework can be customized for any action recognition application. The paper presents the stages of the framework where GPUs is the key for performance improvement. The speedup is achieved in terms of video processing and Machine learning algorithms implemented on GPU. Video processing is supported as GPU implementation of Motion detection and segmentation, beside support of using OpenCV on GPU using GPUCV. Machine learning is provided with implementations of Support Vector Machine (SVM) which is mainly used in object classification and feature discretization, while Hidden Marcov Model (HMM) supports activity recognition phase in the framework. The system was tested against UCF dataset and speedup of 20x has been achieved on NVidia 9500GT graphics card (32 500MHZ processors).

Keywords— *Computer Vision, Action Recognition, GPU, Teamwork.*

1. INTRODUCTION

GPU has been the state-of-the-art technology used to solve performance problem due to its intensive computational power, beside its low cost that leads simple personal computer to have the name of personal super computer by attaching GPU chip. One of the main companies leading this chip industry is NVidia. NVidia presents an architecture known as CUDA provided with SDK that is accessible to software developers through standard programming languages. In the same direction, OpenCL initially developed by Apple Incorporation as standard framework for writing Kernels on heterogeneous platforms consisting of CPUs and GPUs. As a result, advances in GPU industry and development have been achieved and a lot of work has been done to get GPU applied in many high performance fields. Examples include Bioinformatics, Image processing and computer vision.

Applications of image processing include Hyper-spectral image processing [1] where Javier Setoain et al presented an investigation of multi-level parallel implementations. In Computer vision field Jean-Philippe et al presented GPUCV [2] as GPU implementation of popular OpenCV functions. This work has achieved slightly improvement of the speedup in computer vision applications, however optimality is not ensured due to generality of these functions. As a rule, special purpose GPU design results in a big step in performance.

Moreover, Peihua Li et al presented parallel implementation of *mean shift tracking* [3] on GPU as 6 kernel functions. They make use of K-means clustering to partition color space into distribution of small pins; consequently all key components were mapped onto GPU with great speedup.

In the same way, a lot of research has been done towards enhancing performance of detection, tracking and segmentation phases. Furthermore, noticeable effort has been exerted to achieve significant improvement in implementation of machine learning algorithms as a result of its great contribution in various computer vision applications. In this direction, Chuan Lui presented GPU implementations of HMM [4], similarly Austin Carpenter presented GPU implementation of SVM by Austin Carpenter [5].

Currently, GPUs are widely deployed as a powerful resource to achieve intensive computing tasks in high speed. Since Action recognition systems requires huge computational power to fit in real-time. This paper builds a link between GPUs and team work activity recognition systems as a framework on which many real time systems can be implemented. This framework targets the developer of action recognition systems. Through the rest of paper, we will refer to the target users of the framework as developers. This paper is organized as follows. Part II presents the proposed framework. Part III shows High performance approaches in the framework. Part IV shows how the framework could be configured for any system. Part V presents experimental results that report the achieved speedup.

2. GPU TEAMWORK ACTIVITY RECOGNITION FRAMEWORK

For building a channel between GPUs and activity recognition systems, we investigated the major techniques used in action recognition and computer vision having the objective of making this channel serving as general framework on which many applications can be built. Figure1 presents the workflow of the proposed GPU framework. The input is raw images captured of moving camera, and the output is the recognized activity. The colored modules are the ones with general GPU implementation to be configured according to application requirements. The non-colored modules (Agent oriented feature extraction, Team oriented feature extraction) are implemented by the developer according to the potential application running under the framework. However the non-colored modules are implemented by the developer, the framework provides the developer with GPUCV functions which gives him support to achieve speedup with minimal effort. Furthermore, the framework accepts the developer implementation either on GPU or CPU.

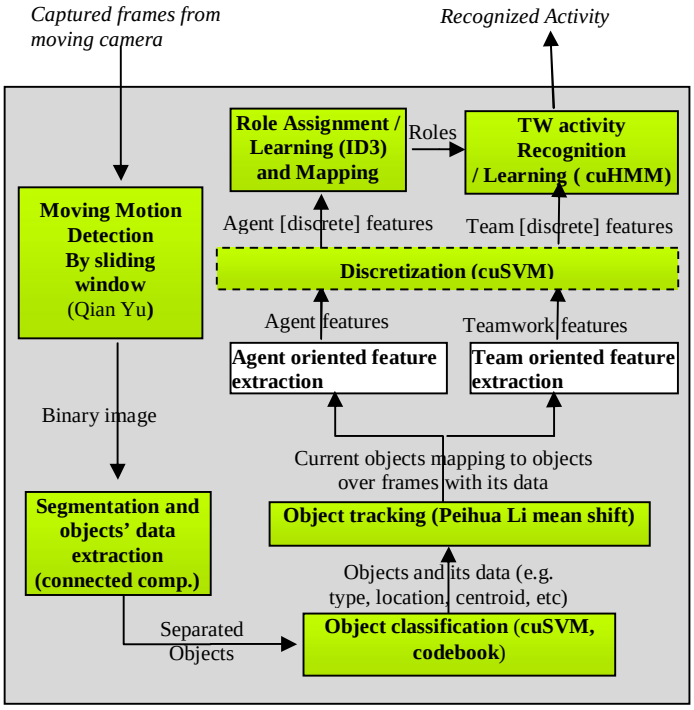

Figure 1: Process flow of the GPU framework

As illustrated in Figure1, the framework spans all stages required for any action recognition system with specs of each component defined on the arrows. Besides, the framework has been built such that functionality of GPUCV (i.e. GPU version of Intel OpenCV functions) [4] is supported and integrated as an infrastructure that can be used by the developer.

Next sections present the detailed implementation of each component in the framework that together comprises the framework.

2.1. Moving Motion Detection by sliding window

Qian Yu et al [6] achieved significant acceleration of motion detection of moving framework based on GPU which leads to building background model and detecting motion regions at 18fps 320x240 videos. The process of detection on GPU was implemented as two steps, warping images and computing the background model. The main objective was to minimize memory transferring between GPU and CPU.

Figure 2 illustrates how the structure stores frames of the window as 2D textures in GPU memory. Next, wrapping which is done on each frame of the window using GPU. Next, output of wrapping is passed to GPU component for collecting statistics from the correspondences of the window which leads to background model.

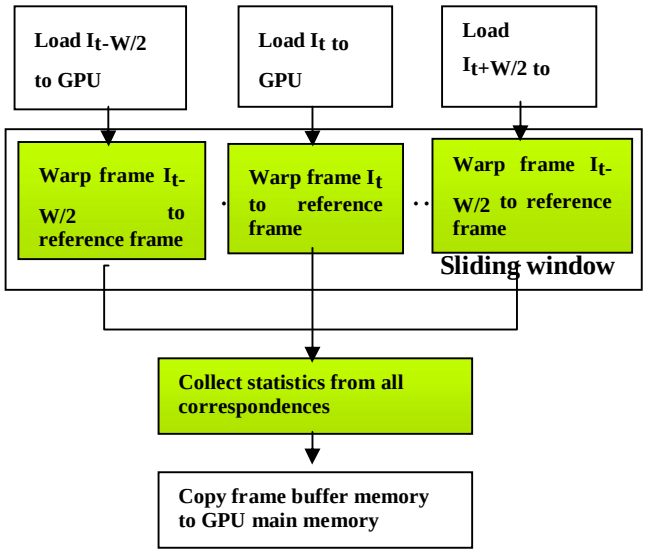

Figure 2: Quan Yu et al Motion detection on GPU.

2.2. Segmentation and objects' data extraction.

In this stage GPU *connected component* is applied to segment objects in the scene (Figure 3). The speedup is gained by dividing image into N blocks then merging the computations as detailed in [7]. But experiments show that the cost of this algorithm depends on the size and complexity of the labeled objects, so this technique may inherently be not faster than the CPU classical 2-passes sequential algorithm achieved by using Intel OpenCV version.

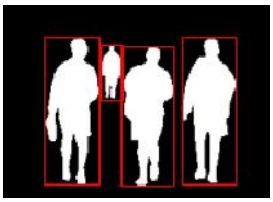

Figure 3: Segmentation and object extraction

2.3. Object classification (cuSVM, codebook)

This stage of processing is responsible for categorizing detected objects into classes. Speedup of this stage can be achieved by multilevel implementation. For instance, Jianpeng Zhou et al presented codebook algorithm to classify humans [8], While Vili Kellokumpu et al used SVM in [9] to classify human postures (Figure 4). So codebook can be used to ensure filtering out non- human objects, then human blobs is presented to SVM to classify human postures. Consequently, GPU implementations of these models are included as a basic part of the framework.

(a) (b)

Figure 4: (a) Classification of object as a ball and human
(b) Classification of different human postures.

2.4. Object tracking (Peihua Li)

In this component correspondence mapping of blobs across frames are constructed. Peihua Li et al's work [3] was based on the idea of partitioning the color space of objects which leads to a quite small number of histogram bins representing color distribution. That histogram was the base component of *mean shift tracking* algorithm. The speedup of object tracking in the framework is achieved by the GPU implementation of that algorithm.

Figure 5: Object tracking

2.5. Discretization (cuSVM)

After the detection and classification of objects, the next is to handle features into HMM for recognition of actions in discrete form which is welcome by HMM because discrete HMM is more accurate than continuous HMM as known in the literature, This phase encapsulate discretization of agent features (e.g. postures into a set of predefined ones, velocity as fast or slow,.., etc), and team oriented features (e.g. cohesion as separated or merged) as illustrated in Figure 5.0.

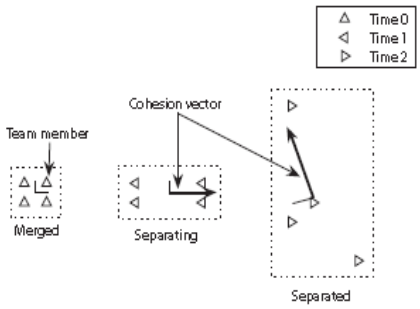

Figure 6: Discrete Team features

In this phase, the framework provides the developer with GPU implementation of SVM to discretize features. Moreover, the developer can integrate his own discretizer into the framework.

2.6. Role Assignment / Learning (C5.0/ID3) and Mapping

In this stage each agent in the team is assigned to a role such that learning phase is done in a consistent and accurate way. Role assignment and mapping is done using C5.0 which is based on ID3 *decision tree learning* algorithm [10]. The classification involves mapping the data structure that describes a decision forest to a 2D texture array. Navigation through the forest for each point of the input data is done in parallel. Regarding training, the responses of the training data are computed to a set of candidate features, then, these responses are scattered into a suitable histogram using a vertex shader.

2.7. TW activity Recognition / Learning (cuHMM)

Activity Recognition is the last phase in the framework that has the function of either learning or recognition according to the mode of the framework. The main Learning model that has been used in action recognition with a great success is HMM. Three kernel GPU implementation functions are provided: *Forward*, *Viterbi* and *BalmWelch* algorithms. HMM supported in this phase is not restricted only to teamwork actions but can be generally used in action recognition phase of any application.

3. HIGH PERFORMANCE APPROACHES IN THE FRAMEWORK

Parallelism of processing for applications under the presented framework can be achieved using the following approaches:

1. Component wise parallelism on GPU in which the processing logic of each colored component in Figure1 is distributed over GPU processing elements.

2. Sequential phases running on the framework can be pipelined (e.g. while detected objects of a frame are segmented in segmentation phase, the motion process is performed in the next frame).

3. Independent task parallelism which can be exploited in object classification and action recognition phases. Object classification is a pure parallel task where multiple objects can be classified simultaneously. In action recognition HMMs are built for each action which can run in parallel.

Methods above show how the parallelism is done. Hence a question appears. Is there another degree of freedom for configuring the hardware to gain more performance improvement? Our investigations led to two types of variations on the hardware platform for accelerating framework task:

1. Multiple GPU tasks can be done by attaching more cards to the same PC if applicable (High speed bus will be critical issue here).

2. The framework can be installed over PCs connected by high speed LAN with each PC attached with one or more GPUs as known as GPUs-Grid.

4. CONFIGURING SYSTEMS RUNNING UNDER THE FRAMEWORK

There are two types of configurations required for full utilization of the GPU framework: one is software and the other is hardware.

4.1. Components' configuration (software)

In this stage, the developer makes selections of stages that will be utilized during the processing of his target system, then adjusts the parameters of each component that fit into system requirements. The configuration parameters for each phase are listed as follows.

- Moving Motion Detection By sliding window: length of sliding windows with 91 as the default length, and number of histogram pins [6].

- Segmentation and objects' data extraction: the number of N partitions [7]

- Object classification: feature vector length, and number of classes [5].

- Mean Shift tracking: the number of clusters K as the tracking algorithm is based on K-means clustering [3]

- Role Assignment and mapping (ID3 algorithm): mode (TreeLeaf, ForestLeaves and etc), number of trees and number of classes [10].

- TW activity Recognition (HMM): number of hidden states, feature vector size, blocks size and number of actions. The default value of the block size is 16 [4].

4.2. Environment setup (hardware configuration)

After components' configuration, the next is to install the hardware components in hand. The minimum requirement is to have one node with one GPU card on which all configured GPU components would run, the first approach in part III.

For more complex situation where environment contains grid of nodes with each node containing at least a GPU card, all types of parallelism approaches in part III are applicable. Pipelining and Independent task parallelism are achieved either within same machine with multiple GPUs or across high speed LAN. The two methods are detailed as follows.

1. Multi-GPUs on a single machine: This is most recommended for tasks that require processing relatively large data to overcome communication cost over intranet. This will be more suitable for pipelining the stages of the configured systems (e.g. detection, tracking, segmentation, etc).

2. GPU Grid distributed over different machines: This is recommended if a lot of learned data is stored over network and hence features would have relatively small size to broadcast over network then gain recognition data. This would be suitable for classification and recognition that use large data for

learning (e.g. object classification, action recognition phases).

The key of successful configuration of the framework is how to organize available resources to fit processing requirements of the target system with optimal GPU utilization The optimum goal here is to configure the framework to minimize the processing time (denoted as T) required for recognizing actions given by Equation 1:

$$T = \sum PTcpu(i) + \sum PTgpu(j) + \sum CTgpu\_cpu(k) + \sum CTnode\_node(l) \quad (1)$$

Where $PTcpu(i)$ is the processing time of task i on a CPU. $PTgpu(j)$ is the processing time of task j on a GPU. $CTgpu\_cpu(k)$ is the communication time for task k between a GPU and a CPU. $CTnode\_node(l)$ is the communication cost for task l between two nodes belonging to the GPUs-Grid if applicable.

5. EXPERIMENTAL RESULTS

This section provides speedup results for each of the GPU components comprising the framework. Besides, it provides overall speedup of the framework in terms of a selected application.

5.1. Moving Motion Detection by sliding window

GPU version using the adaptive *mean* as the background model with 91 frame sliding window can run at around 18 fps on 320x240 resolution videos. The *mode* approach can run at 10 fps with the same setting. The GPU version of the *mean* approach achieves around 12x speedup over its standard CPU counterpart. The GPU version of the *mode* approach achieves around 15x speedup.

5.2. Segmentation and objects' data extraction

This part is not recommended to use due to its GPU version is slower than CPU version. In case of image with 2048x2048 as dimensions on 9800GT, the speedup is 0.769231x which means slowdown, however it could result in speedup if GPU with higher capability is used. So it is subject to use depending on HW and complexity of input objects.

5.3. Object classification and Discretization (cuSVM)

Achieved speedup in cuSVM reached about 35x for training SVM, 84x for corresponding prediction. This speedup was recorded against MNIST data set (LeCun et al., 1998) on NVIDIA GTX 260 GPU.

5.4. Object tracking

The experiments show a speedup in blob tracking reaching 3.36x using the famous CAVIAR data set on GeForce 8800 GTS GPU.

5.5. Role Assignment / Learning (C5.0/ID3) and Mapping

129x speedup in argmax mode using object recognition as an application on GPU nVidia GeForce GTX 280 (240 stream processors)

5.6. TW activity Recognition / Learning (cuHMM)

880x speedup of forward algorithm GPU implementation while 180x is the speedup of Baumwelch algorithm GPU. This test was done on 512x512 (number of states x number of sequences) on NVIDIA G92 with 512MB RAM.

5.7. Evaluation of framework speedup

To test the applicability of the framework, we selected the system in [11] which is "role based activity recognition in observations of embodied agent actions".

The researchers in [11] used labeled data set, so motion detection, segmentation and tracking are filtered out in components' selection step of the configuration. The components' configuration of that system is presented as follows.

- Discretization: in this component, the developer uses custom thresholds to transfer features from continuous to discrete.
- Role Assignment and mapping (ID3 algorithm): TreeLeaf mode, single tree and 4 classes (4 different roles).
- TW activity Recognition (HMM): number of hidden states is five, feature vector size is 6, and number of actions is six. .

This system was tested using UCF dataset on NVidia 9500GT graphics card (32 500MHZ processors) against 3.2GHZ dual core processor, and speedup of 20x was gained. Gained speedup increases dramatically as capabilities increases.

6. CONCLUSION AND FUTURE WORK

We have presented a GPU framework for action recognition that can be configured to any action recognition system, furthermore key approaches for good utilization of the framework have been presented. Experiments shows how the framework is configured in teamwork action recognition system with speedup of 20x achieved on NVidia 9500GT graphics card.

Future work includes enrichment of each phase on the framework with more GPU implementations of relevant algorithms .Examples include GPU implementations of

- Particle filters for object tracking.
- Correleogram based segmentation algorithm to resolve the problem of overlapping objects.
- Codebook based algorithm in object classification phase.

Additionally a study can be build up on the framework showing how it varies with different setups of GPU-Grids with variation of number of nodes in the Grid and number of GPU cards in each node. This study may lead to automatic configuration of any system according to the available hardware.

7. REFERENCES

[1] Javier Setoain, Manuel Prieto,Christian Tenllado ,and Francisco Tirado, "GPU for Parallel On-Board Hyperspectral Image Processing", Sage Publications,CA,USA, November 2008 .

[2] Jean-Philippe Farrugia, Patrick Horain, Erwan Guehenneux,and Yannick Alusse, "GPUCV: A FRAMEWORK FOR IMAGE PROCESSING ACCELERATION WITH GRAPHICS PROCESSORS", Lyon University,2006.

[3] Peihua Li and Lijuan Xiao, "Mean Shift Parallel Tracking on GPU", Springer-Verlag Berlin, Heidelberg,2009.

[4] Chuan Liu, "a CUDA Implementation of Hidden Markov Model Training and Classification" Johns Hopkins University, May 2009

[5] Austin Carpenter,"A CUDA IMPLEMENTATION OF SUPPORT VECTOR CLASSIFICATION AND REGRESSION",2009

[6] Qian Yu, and Gerard Medioni, "A GPU-based implementation of Motion Detection from a Moving Platform", IEEE ,23-28 June 2008.

[7] Sean M. O'Connell , "A GPU Implementation of Connected Component Labeling ", Illinois State University, 2009.

[8] Jianpeng Zhou and Jack Hoang , "Real Time Robust Human Detection and Tracking System" San Diego, USA , IEEE, 2005.

[9] Vili Kellokumpu, Matti Pietikäinen, and Janne Heikkilä. "Human Activity Recognition Using Sequences of Postures",p 2 . IAPR, 2005.

[10] Toby Sharp, "Implementing Decision Trees and Forests on a GPU", Springer ,2008.

[11] Linus J. Luotsinen and Ladislau Bölöni "Role-Based Teamwork Activity Recognition inObservations of Embodied Agent Actions", Richland, SC ,2008