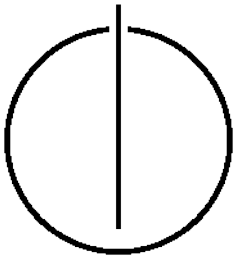
TECHNISCHE UNIVERSITÄT MÜNCHEN

DEPARTMENT OF INFORMATICS

Guided Research

Rajat Koner



TECHNISCHE UNIVERSITÄT MÜNCHEN

DEPARTMENT OF INFORMATICS

Guided Research

Multi-Pedestrian Tracking with GM-PHD Filters in an embedded Heterogeneous Parallel Processing Platform with Sensor Fusion

Author:	Rajat Koner
Supervisor:	Biao Hu
Advisor:	Dr.Kai Hung
Submission Date:	18th Sept 2016

I confirm that this guided research is my own work and I have documented all sources and material used.

Munich, 18th Sept 2016 Rajat Koner

Abstract

Real-Time Pedestrian detection and tracking methods are the state-of-the-art techniques in present driver assistance systems. However, pedestrian detection and tracking methods that exploit the parallel processing capabilities of heterogeneous high performance computing devices such as FPGAs (or GPUs) with sensor fusion(camera and Lidar), a technology that potentially will replace ECUs in a coming generation of cars, are a rare subject of interest. In this research a pedestrian detection and tracking algorithm is developed and implemented, especially designed to incorporate one or many, and even heterogeneous, hardware accelerators in the first phase. In the second phase it will incorporate Lider and use the data fusion technique to gain better accuracy and precision in real-time. Pedestrian detection is done using Histogram of Oriented Gradients (HOG) for human detection. Parallel implementation of HOG people detection had given a very good real-time performance and robustness. For tracking pedestrian we have used Guassian Mixture of Probability Hypothesis Density filter, which is very suitable for sensor fusion because of its output set based. The code of the parallel implementation of HOG and sequential implementation is tested on 2 GPU – the NVIDIA GeForce 940 M and NVIDIA GeForce GTX 660 TI. Tests on GPU show a significant improvement in performance and memory consumption. Detection performance is improved by 10x to 15x on an average by these above mentioned GPU. The algorithm is tested with different benchmark dataset and performed very well with speed and accuracy. Sequential implementation of GMPHD filter performance heavily decreased with increasing number of target. Parallel implementation of some of the block of GMPHD filter achieved 10x performance speedup.

Contents

1 Introduction

In this paper, a novel pedestrian detection and tracking method is developed and specifically designed for heterogeneous hardware accelerators and sensor fusion. In pedestrian detection HOG based human detection is a very widely used and robust approach , though HOG is very resource consuming and slow specially for large image. Through the parallel implementation of it overcame this shortage and achieved a very good performance enhancement which is suitable for a real-time system. For tracking we are using GM-PHD filter, which gives a comparatively good performance but used to decrease with an increasing number of target and memory consumption is very high. In future, we will integrate both car and pedestrian detection in a single frame, so the number of targets will increase. So for large scale and real-time scenario in an embedded system is not good. GM-PHD filter consists of three main part prediction, update, and pruning. So parallelizing each block will greatly improve its performance and memory consumption. And incorporating it with Lider improves overall accuracy and its reliability on different weather and environmental condition. Combining camera and Lider for this and having a fast parallel algorithm is very suitable and preferred approach and a new contribution in this field.

1.1 Structure of the Method

In the first step, the algorithm pre-processes the incoming frames to emphasize region of interest. Then the algorithm proceeds for the pedestrian detection. This algorithm may take two different ways, either it would go for tracking the detected target in the frame or detection and tracking frame. If targets are detected and predicted position of the target matches with the actual detection, a less expensive GM-PHD tracking will be used for the tracking of that pedestrian. The latter is applied in the vast majority of the frames.

The computation in these two stages pedestrian detection and pedestrian tracking can be partly or completely performed in parallel. The respective parts are implemented in the form of OpenCL kernels and are executed on a hardware accelerator. While GM-PHD filter parallel implementation will incorporate specialized vectorization technique to enhance frequent small 4x4 or 4x2 matrix operation .

1.2 Basic FLowchart

Below diagram shows the different block and its basic operation for a fast implementation , it also shows algorithm of this overall process . As target detection is a very expensive process ,so we can skip target detection for some frame and can rely on tracker output.As after some frame when tracker stabilize we can calculate speed and position of the target.This way we can improve system performance without reducing accuracy.

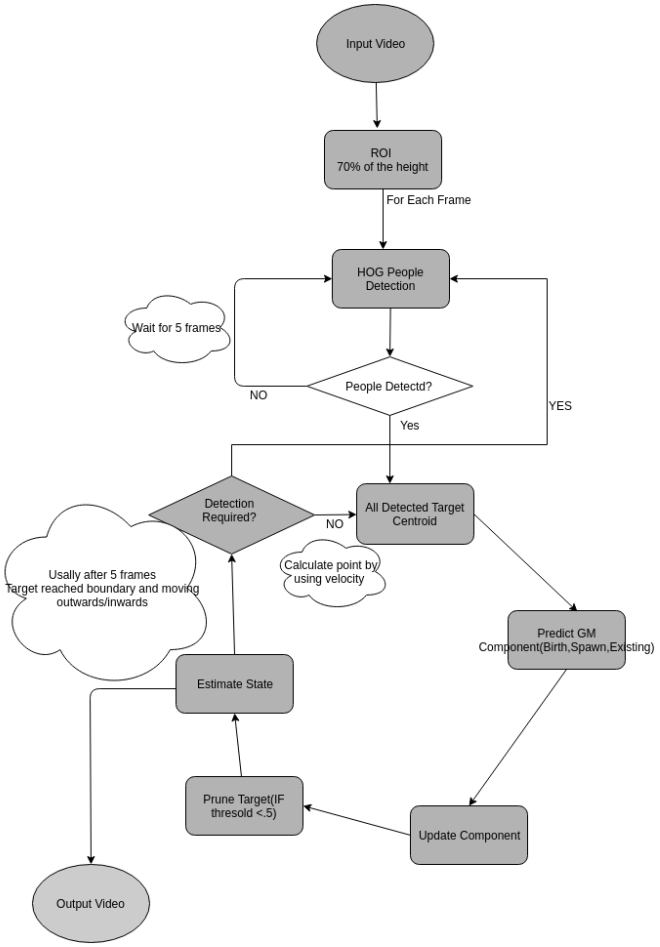


Figure 1.1: Basic Sequence diagram .

2 Implementation

HOG based descriptor is very widely used for both pedestrian and vehicle detection technique.Though several new improvements for this technique has been proposed to increase accuracy which considers local preferences as well as the global descriptor.One of the most beneficial aspects of this global descriptor can be used to detect other objects present in the frame like Car.

2.1 Pedestrian Detection using HOG

The first part of this is intended to have a good real-time pedestrian detection algorithm based on which tracker will do the tracking. For detection Human "Histogram Oriented Gradient detection of Human" is a very widely used approach. HOG is a global descriptor, it provides a description of the entire frame. It's divided entire frame into the 2x2 block and compute gradient orientation and magnitude. Then it divides 9 bins from 0-180 degree and concatenates it in a 1D feature vector. This feature vector feeds into a pre-trained Support Vector machine for detecting human. But this algorithm run very slow near 0.7FPS, which is not at all applicable to a real-time system.

2.1.1 HOG Extraction

- Compute centred horizontal and vertical gradients with no smoothing.
- Compute gradient orientation and magnitudes For color image, pick the color channel with the highest gradient magnitude for each pixel .

2.1.2 For a 64x128 image

- Divide the image into 16x16 blocks of 50 percent overlap. 7x15=105 blocks in total .
- Each block should consist of 2x2 cells with size 8x8.
- Quantize the gradient orientation into 9 bins
- The vote is the gradient magnitude

- Interpolate votes between neighbouring bin center.
- The vote can also be weighted with Gaussian to down weight the pixels near the edges of the block.
- Concatenate histograms (Feature dimension: 105x4x9 = 3,780)
- 7x15 =105 blocks in total

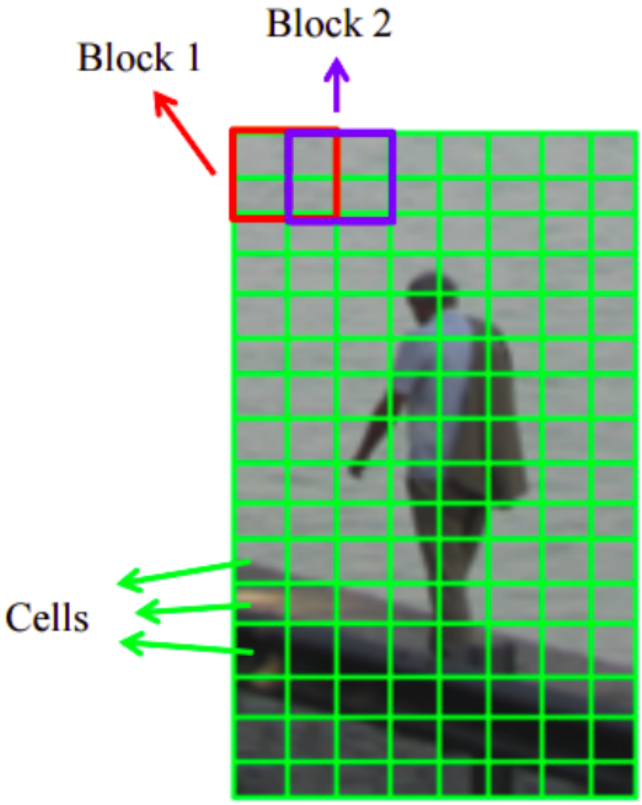


Figure 2.1: Half overlapped 16x16 bolck.

Votes

- Quantize the gradient into 9 bins(0-180)
- The vote is the gradient magnitude

- Interpolate votes between neighbouring bins center.
  1. Example :If degrees ,then distance from bin center 70 and 90 are 15 and 5 degrees.
  2. Hence ratios are 5/20 = 1/4 and 15/20 = 3/4
- Vote can also be weighted with Gaussian to down weight the pixel near the edges of the block

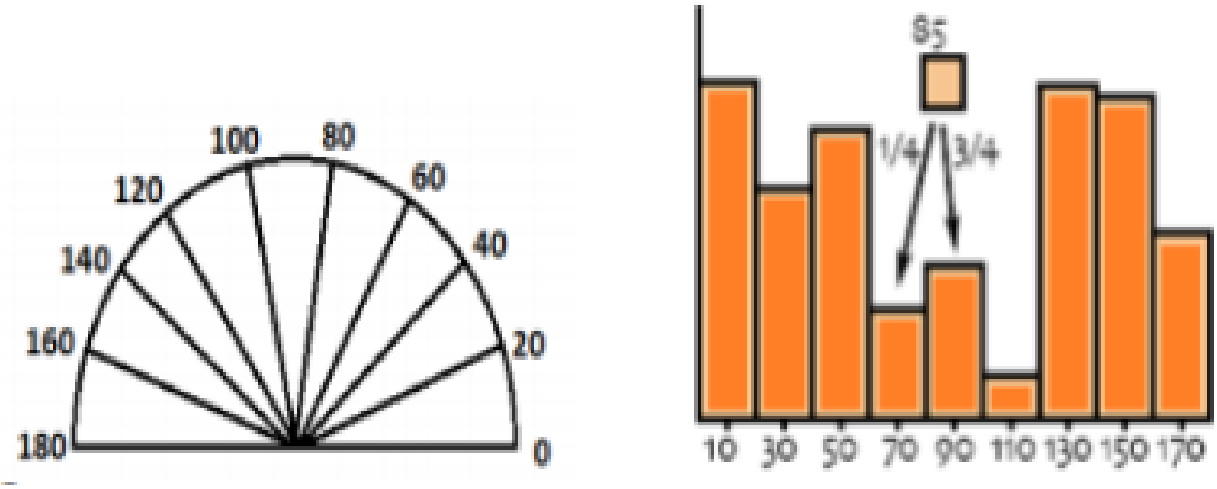


Figure 2.2: 9 Bins and center of Bins.

Final Feature Vector

- Concatenate histogram
- Make it a ID vector of length of 3780
- Feed Data into a Support Vector Machine

These final histogram can be visualize as below.Then this feature vector can be feed to some pre trained Support Vector Machine and gives the position of human detected in frame .

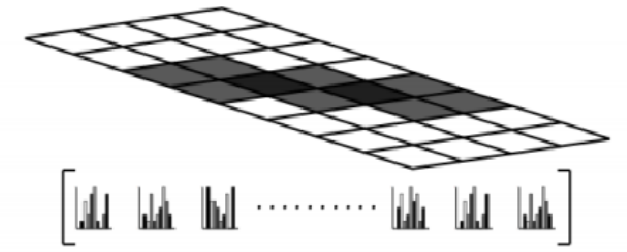


Figure 2.3: Concatinated histogram.

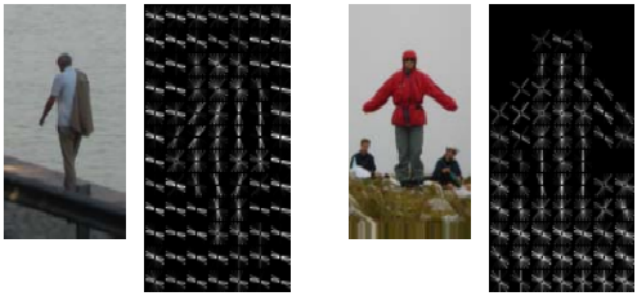


Figure 2.4: Histogram visualisation of HOG.

2.2 Parallel Implementation of HOG

In the histogram computation step, a HOG pixel block is mapped to a thread. In the case of pedestrian detection the block has 4 cells and each cell 8 columns of 8 pixels, so we will use 8x4=32 threads. In our implementation, each thread computes its own histogram and stores it in shared memory. This block size imposes some certain restriction on the local memory block size. After completing all the histogram calculation these are merged using parallel reduction technique. In this step, each detection window is mapped to a OpenCl thread block. In our case a detection window has 64x128 pixels, so it will be made of up 7x15 blocks. Then each position and all scales the SVM scores are stored in global memory and then transferred back to host memory. This is done using OpenCV Ocl api.

After this parallel implementation we can achieve a speed of 40x in NVIDIA GeForce 940M GPU and 45-50x GeForce GTX 660 TI. This speedup is really suitable for a real time system.

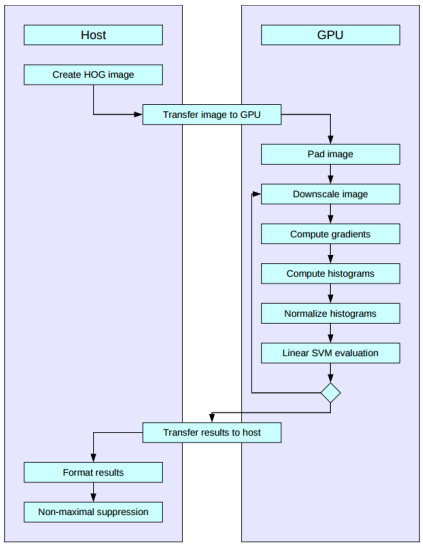


Figure 2.5: Block diagram of GPU implementation of HOG.

2.2.1 Parallel HOG Output

Figure 2.6: HOG pedestrian detection.

HOG implementation is computationally very expensive and slow process.HOG algorithm also suffers from various environmental issues such as shadows and partially visible human and it impacts directly on its accuracy.Though it provides a really and robust base for detecting human and other objects.

2.2.2 Challenges

HOG human detection is quite good for most of the time ,but presently there are very good human detection technique available, one of them is the extension of this is called DPM based technique which also used global HOG as a descriptor and consider many of its local feature. In future we can extend this for better accuracy . Also for car detection we can use the same HOG descriptor.

2.3 Gaussian Mixture PHD filter

In an embedded system like ECU where performance, memory and power consumption all are critical trade-off, in a system like that GM-PHD filter is a good solution for pedestrian tracking. GM-PHD filter is widely used as a multi-target tracking scenario where object follows linear Gaussian model. But it also performs well in other scenarios, where target follows non-linear motion. GM-PHD filter is computationally not so expensive compared to SMC version which is supposed to be better in dealing with non-linear motion and also it provides better estimate most of the case. Though overall performances of GM-PHD filter is good but increasing number (>10) of target reduces GM-PHD filter performance very drastically both in execution time and memory consumption. In real-time scenario where lot of crowd moves or in worst cases it can take a lot of CPU and other resources for execution. In real time system it is a bottleneck. However an optimize GPU implementation can overcome this, it can improve performance and low power consumption.

2.3.1 GM-PHD Filter Basics

GM-PHD filter is an analytic solution to the PHD filtering for linear Gaussian target dynamics and Gaussian birth model. The GM-PHD propagates the Gaussian mixture posterior intensity in time using the PHD recursions. On the contrary to particle based approaches, the GM-PHD provides reliable state estimates extracted from the posterior intensity in a much more efficient way. The GM-PHD filter in its original form only provides the state estimate of individual targets as a set-valued estimate of the multi-target state and the estimate of the target number .Which is ideally very suitable for sensor fusion.

The GM-PHD filter estimates multi-target states by determining the Gaussian components with highest/threshold weights. There are several assumptions used in the GM-PHD recursions.The main three assumptions, are as mentioned below :

- Each target follows a linear Gaussian dynamical model, i.e.

$$F_{k|k-1}(X|\zeta) = \mathcal{N}(x; F_{k-1}\zeta, Q_{k-1}).$$

$$g_k(z|x) = \mathcal{N}(z; H_k x, R_k).$$

where $\mathcal{N}(;m,P)$ denotes the Gaussian density with mean m and covariance P. F_{k-1} is the state transition matrix and Q_{k-1} is the process noise covariance, H_k is observation matrix, and R_k is the observation noise covariance.

- Secondly, the detection and survival probabilities are state independent: pD,k(x) = pD,k .

- Lastly, the intensity of the birth RFSs is Gaussian mixtures of the form

$$\gamma_k(x) = \sum_{n=1}^{J_\gamma,k} \omega_{\gamma,k}^{(i)} \mathcal{N}(x; m_{\gamma,k}^{(i)}, P_{\gamma,k}^{(i)}).$$

2.3.2 Equation for GM-PHD filter

$J_{y,k}$ and $W_{(i)}^{\gamma,k}$ are given model parameters that determine the birth intensity.Posterior intensity at time $(k-1)$ can be written as a sum of Gaussian components with different weights, means and covariances as

$$\nu_{k-1}(x) = \sum_{n=1}^{J_{k-1}} \omega_{k-1}^{(i)} \mathcal{N}(x; m_{k-1}^{(i)}, P_{k-1}^{(i)}).$$

At time k, the predicted intensity is also a Gaussian mixture:

$$\nu_{k|k-1}(X) = \nu_{S.k|k-1}(X) + \gamma_k(X).$$

where ,

$$\nu_{S.k|k-1}(X) = p_{S,k} \sum_{j=1}^{J_{k-1}} \omega_{k-1}^{(i)} \mathcal{N}(x; m_{k-1}^{(i)}, P_{k-1}^{(i)}).$$

The posterior intensity at time k is also a Gaussian mixture and can be written as :

$$\nu_k(x) = (1 - p_{D,k})\nu_{S.k|k-1}(X) + \sum_{z=Z_k} \nu_{D,k}(x,z)$$

$$\nu_{D,k}(x,z) = \sum_{j=1}^{J_{k|k-1}} \omega_k^{(j)} \mathcal{N}(x; m_{k|k}^{(j)}, P_{k|k}^{(j)}).$$

$$K_k^{(j)} = P_{k|k-1}^{(j)} H_k^T (H_k P_{k|k-1}^{(j))} H_k^T + R_k)^{-1}$$

In update step calculation is required for $q_k^j(Z)$ as weight ,$m_{k|k}^{(j)}(z)$ mean and $P_{k|k}^{(j)}$ as covariance for all component and new target.

$$q_k^j(Z) = \mathcal{N}(z; H_k m_{k|k-1}^{(j)}, R_k + H_k P_{k|k-1}^{(j)} H_k^T).$$

$$m_{k|k}^{(j)}(z) = m_{k|k}^{(j)} + K_k^{(j)}(z - H_k m_{k|k-1}^{(j)}).$$

$$P_{k|k}^{(j)} = [I - K_k^{(j)} H_k] P_{k|k-1}^{(j)}$$

With every iteration number of Gaussian Component increase which incur a lot of computational problems. To get rid of this a simple pruning and merging procedure is used to limit the number of Gaussian elements to be propagated in the next label. Firstly, weights below a predefined threshold are eliminated. Moreover, some of the Gaussian components are so close together that they could be accurately approximated by a single Gaussian. Hence, in practice these components can be merged into one.

2.3.3 Output of GM-PHD filter

Figure 2.7: GM-PHD filter on ETH Zurich Dataset

GM-PHD filter theoretically is meant for a target which follows linear motion.In practice or in this output works quite well for a pedestrian who followed random or non-linear motion.It associates two closely related targets as one.This feature ideally very suitable for sensor fusion .

2.4 Parallel Implementation of GM-PHD filter

2.4.1 Introduction

GM-PHD filter actually contains three block prediction ,update and pruning and each block contain series of matrix operation which are addition, multiplication, inverse and transpose. This three block are sequential and cannot be parallelized. However after analyzing the execution path, parallelizing of those matrix operations is possible. But all these matrix dimensions are very low usually 4x4. So creating a parallel implementation of this matrix operation will not be useful as HOST to GPU memory transfer is comparatively very slow. This reduces to only one option we can create a thread for each Gaussian component and take advantage of OpenCL highly optimized vector processing.

As most of the matrix equal or less dimension than 4x4 then using of inbuilt OpenCL vector like **float16** or **float4** can give a good performance enhancement . All these vectors can perform bulk load and read from memory .To get values from this vector optimally we have to use reduction technique.

2.4.2 Vectorization of Kernel Code

GM-PHD filter has three main block,each of them contain multiple small matrix operation and some adjustment and normalization of weight of a Gaussian component .As number of detection increase these component increase quadratically in combination of Target Birth,Target Spawn and surviving target of previous iteration.Calculation of covariance ,mean and weight of all these Gaussian component done in OpenCL kernel.Below is the list of matrix operation need to be done at each block . Below is some sample of vector code.

```
// transpose a 4x4 matrix
void transpose( float4 m[4] )
  {
   float16 x = (float16)( m[0], m[1], m[2], m[3] );
   float16 t; //transpose
     t.even = x.lo; t.odd = x.hi; x.even = t.lo; x.odd = t.hi; //write back
       m[0] = x.lo.lo;
       m[1] = x.lo.hi;
       m[2] = x.hi.lo;
       m[3] = x.hi.hi;
  }
```

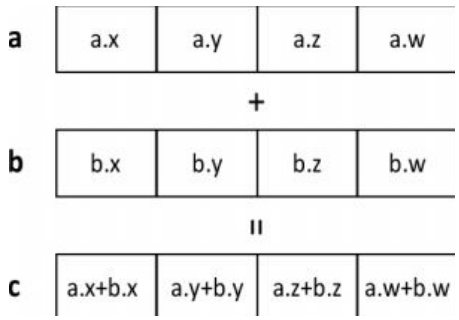


Figure 2.8: Vetor addition .

Parallel Reduction A reduction operation is any operation that takes a vector and reduces it to a scalar, one example reduction is the sum-of-all-elements-operation. It is trivial to implement a sum of all elements in a sequential fashion; simply iterate over all values and add them one at a time to an accumulator variable.

We use the dot product to get scalar value from a vector. This vectorization and reduction take advantage of locality and associativity and can be at most 15x faster than naive code .

2.4.3 Output of Parallel GM-PHD filter

While driving in cities ,highways or in other roads many challenging scenarios may occur where detection of pedestrian may not be continuous,accurate or false detection from shadow or human-like object.Also, a sudden increase of a number of people reduces tracker performance in a drastic manner .Hence it's very important for the tracker to provide a consistent,fast and reliable tracking in all these scenarios.So below we present a comprehensive performance of GM-PHD filter in various challenging scenario . Our implementation of GM-PHD filter provides a consistent time bound and memory efficient application in all below mentioned scenario .

Figure 2.9: Linear Dynamic target with static target

In this video[2.9] most of the people following a straight line motion while an old man in the left side of the frame standing at a position . It can be seen that GM-PHD filter able to track all the targets consistently.

Figure 2.10: Linear and Mixed motion target.

In this video[2.10] most human following linear motion and few are following random motion.Although multivariate Gaussian mixture only follows linear motion,but in practice, it works well for random motion too.

Figure 2.11: Targets with clutter and inconsistent detection.

This video[2.11] contains some clutter ,and target which are very far from camera,so their detection misses in some frame .This implementation provides very good output even if frame consists clutter and inconsistent detection.

Figure 2.12: Large number of target with random motion.

This video[2.12] contains large number of people whose motion are totally random . Parallel implementation of it provides around 10x faster speed and able to produce a satisfactory result.

2.4.4 Overall Speed Up

To compute overall speed up and comparison between host and GPU we used a video sequence of 600 frames used in figure [2.10].This parallel implementation able to produce around 10x faster result compares to its naive implementation.As it uses vectorized computation as per OpenCL standards ,so its performance will improve a bit with any AMD GPU and optimization of block size based on different architecture and its size of the thread block.

With an increasing number of target performance of GM-PHD filter deteriorate very rapidly.For a futuristic purpose ,where GM-PHD filter to be used for tracking a large number of targets like both pedestrian and car along with other detected object the performance of this filter may not be so efficient to use real-time application.

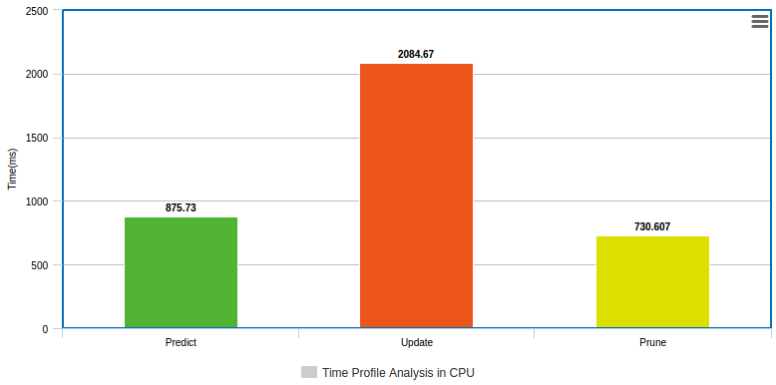


Figure 2.13: Total computation time of different block in host.

As figure 2.13 shows the distribution of computation of different block,where update block consumes the most amount of time.Careful parallelization of these three block can increase the filter's speed and reduce a considerable amount of memory usages.Parallelizing update and prune block are tricky as both blocks contain several normalizations and sequential delete operation which requires the lock and can greatly reduce the performance .

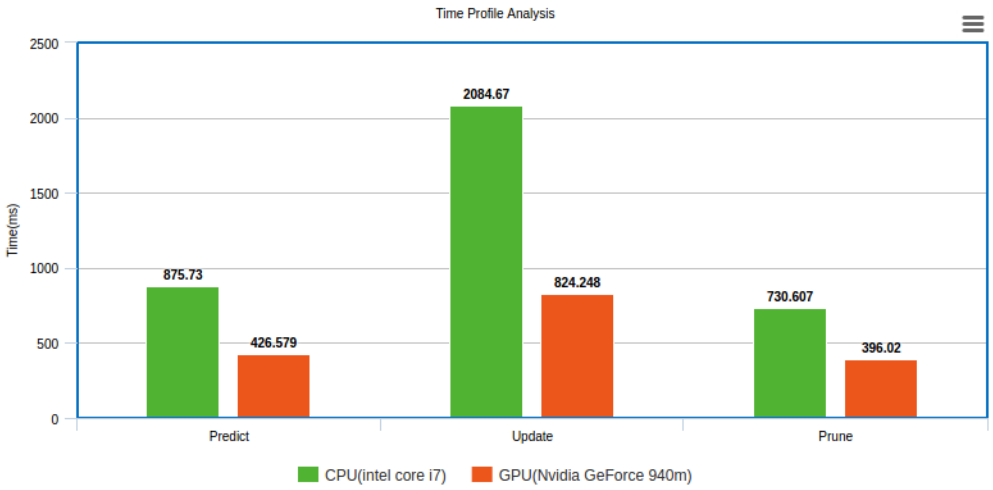


Figure 2.14: Comparision of CPU vs GPU computation time.

This performance will increase if we increase the number of target till a certain amount,and performance will decrease with number of target.In below figures we give an example of this.

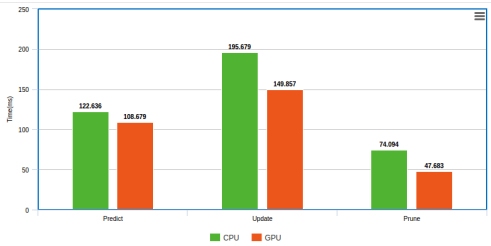


Figure 2.15: Speed up for fig 2.9 with an average of 5 target.

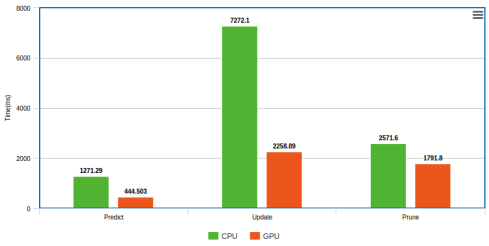


Figure 2.16: Speed up for fig: 2.11 with an average of 13 target .

In above figure we can see the speed up for two different kind of scenario,where number of target is less then speed up is almost equal to CPU but it increases with number of target and it will increase even more if we increase more number of target.And also some optimization technique like setting local thread block size, kernel optimization and OpenCL vectorized code gets special support from AMD GPU can enhance further increase of performance around 10-15x speed-up.

2.4.5 Distribution of Computation

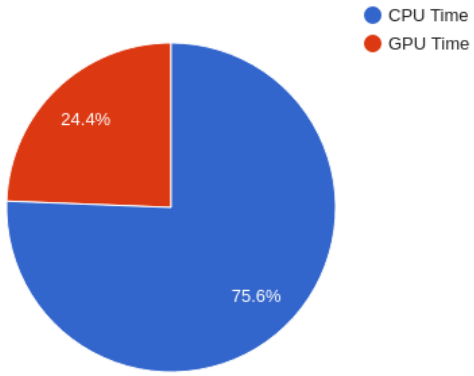


Figure 2.17: Distribution of comutation time between CPU(intel core i7) and GPU(Nvidia GeForce 940M)

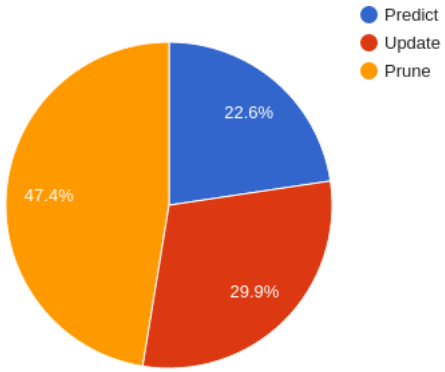


Figure 2.18: Distribution of comutation time between Predict,Update and Prune block.

In a setup using a video of 600 frames with a resolution of 1920x1080 pixels per frame and around 23 target scenario only tracker algorithm requires 1.98 ms average time per frame on intel core i7 CPU and Nvidia GeForce 940M GPU (fig: 2.17).It was expected CPU will consume more time than GPU,as much of sequential operation like normalization of weight,delete is performed on the host side . We consider copying of data from the host to GPU as a part of distributive computation time on GPU.

In figure 2.18 Prune block consumes the most amount of time because of its check a large number of the weight value and delete them based on their proximity to high weight target as they represent the same target.Also, some part of update and predict block can't be parallelized as some part requires the accumulation of previously calculated weight and some block computes a very small number of data which is not worthy to compete in GPU.

2.5 Future Work

In the 2nd phase, we will incorporate Lider and camera for better accuracy and precision and explore system full potential with heterogeneous parallel programming to get a real time system which can be used like an embedded system like ECU. The main focus of this 2nd phase development will be to detect both car and pedestrian in all

environmental condition .Now car and pedestrian motion follow the different model of dynamics ,as mostly car motion are linear while pedestrian uses non-linear motion. To handle this problem we will also use Multi Bernoulli Particle filter.

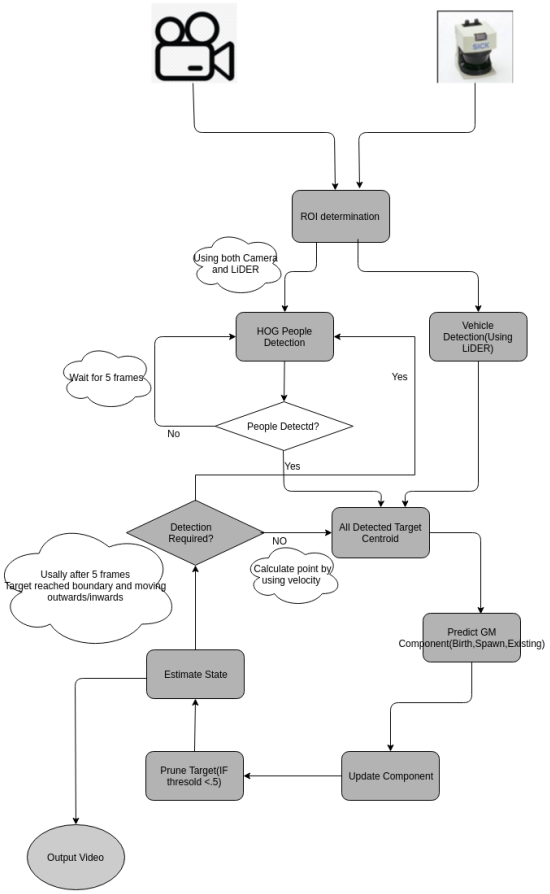


Figure 2.19: Block diagram for future implementation.