Chapter 6

Parallelisation of Multi object Tracking On GPU

6.1 Introduction

Advancement of modern 3D graphics processor facilitate researchers to write parallel codes for their serial programs. GPU’s have more computational power than CPU’s and are fine tuned with parallel codes. GPUs are streamed processing model which uses thousands of stream processors with a fast interconnected network and dedicated pipeline. CPUs have larger caches with high band width that facilitate reuse of data and become suitable for the general purpose computation in contrast to GPUs which have much smaller cache with lower bandwidth with large number of processing elements. This makes graphics application run much faster. General purpose applications are ported to the GPU kernels to create massive thread level parallelism (TLP), which is then exploited by GPUs to concurrently execute thousands of task. Multi object detection and tracking using covariance descriptor involves with lots of computational operations and memory. Those computational intensive operations can be executed at a faster rate by applying group of independent instruction in the streamed processor of the GPU.

In continuation to the last chapter, In this chapter we have developed
multi object tracking using orthogonal matching pursuit algorithm which is implemented on GPU environment. Multi threaded level parallelism is exploited by submitting the independent operation of the tracking algorithm in the kernel of the GPU.

In this work, multi object tracking is carried out with detection and classification. Background subtraction is applied for the detection of the objects. In classification, each object are classified by matching the object with the corresponding trained object of the dictionary. New objects appear for first time are assigned to the dictionary with a distinguish class label. Orthogonal matching pursuit (OMP) algorithm is a sparse based classification algorithm which is used for the classification of the object.

**Problem definition:** Detection and tracking the multi object in the video frame by exploiting the multithread parallelism on GPU platform to reduce the computation time.

**Objective:** Our proposed model has been developed to achieve the following two key objectives.

- To enhance the parallelism by exploiting multiple thread of execution for detection and tracking the multi object.
- To improve the tracking accuracy by searching the target in the given search area in a faster way with the help of parallelism and also remove the false detection of the target in the frame.

**Contribution:** As compare to existing method described in the literature survey, contribution of the paper are as follows

- Our tracking model use covariance descriptors for representing the objects that increase the accuracy of the object classification. At regular interval, a new updated feature descriptor is formed from the set of existing feature descriptor which belongs to a class and that is added to the dictionary to increase the accuracy of the classification.

This chapter is organized into six sections. In the next section 6.2 we present the detection and representation of the objects. Sparse representation
Based classification is described in the section 6.3. The proposed parallel omp algorithm along with its implementation details is presented in section 6.4. Performance analysis and comparison with serial algorithm are discussed in section 6.5 and in the last section we have discussed the conclusion 6.6.

6.2 Object Detection and Representation

Moving object can be identified in the target frame by background subtraction method [36] along with image filtering operation. Each of the detected object is represented with the bounding box. Using Tuzel et al. [8] covariance based representation, pixel present in the bounding box is convert into the feature matrix. i.e bounding box for the image regions are converted into the feature matrix using equation (6.1)

\[
f_k = [x, y, I(x, y), I_x, I_y, \sqrt{I_x^2 + I_y^2}, \arctan \frac{I_x}{I_y}] \quad (6.1)
\]

Here, x and y represent the coordinate position of the pixel in x and y direction respectively. I(x,y) is the intensity value of the pixel I_x and I_y is the derivative of the pixel in x and y direction respectively. As shown in the Figure 6.1(f-h), Each object present in the scene of size m x n is converted into the feature matrix \( F_t \) of size m x n x d. This feature matrix \( F_t \) is further transferred into a low dimensional covariance matrix of dimension d x d and then rewritten into a vector \( a_{tj} \) by reading the elements of the covariance matrix column wise.
Figure 6.1: Proposed scheme for object tracking, (a)-(e) shows the background subtraction for detection of objects, (f)-(h) shows the representation of object with covariance descriptor.

\[
\text{Cov}(F_t) = \begin{pmatrix}
    c_{t,11} & c_{t,12} & \cdots & c_{t,1d} \\
    c_{t,21} & c_{t,22} & \cdots & c_{t,2m} \\
    \vdots & \vdots & \ddots & \vdots \\
    c_{t,d1} & c_{t,d2} & \cdots & c_{t,dd}
\end{pmatrix} = \begin{pmatrix}
    c_1^t \\
    c_2^t \\
    \vdots \\
    c_d^t \\
    \vdots \\
    c_{d+1}^t \\
    \vdots \\
    c_{d^2}^t
\end{pmatrix} = a_t
\]

Finally, vector \( a_t \) of size \( d^2 \times 1 \) represent the object present in the scene.
6.3 Sparse Representation Based Classification

Sparse approximation is one of the popular methods employed for classification problems. This method is used for classifying the object with reduced cost and low memory. This algorithm can be expressed as follows:

\[
\begin{pmatrix}
  y_1 \\
  y_2 \\
  \vdots \\
  y_m
\end{pmatrix} =
\begin{pmatrix}
  a_{11} & a_{12} & \cdots & a_{1m} \\
  a_{21} & a_{22} & \cdots & a_{2m} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{m1} & a_{m2} & \cdots & a_{mn}
\end{pmatrix}
\begin{pmatrix}
  x_1 \\
  x_2 \\
  \vdots \\
  x_m
\end{pmatrix}
\]

In sparse-based classification, we have provided input test sample \( Y \in \mathbb{R}^N \) and the dictionary \( A \in \mathbb{R}^{M \times N} \) and need to find a sparse vector \( X \) that gives the approximation solution \( Y \simeq A \times X \).

In our multi-object sparse classification model, we have considered \( y \in \mathbb{R}^{d^2} \) be the test sample and \( A \in \mathbb{R}^{d^2 \times N} \) is the dictionary that keeps all the object descriptors that are collected from the previous frame. Here \( d^2 \) represents the size of the feature covariance descriptor for an object and \( N \) represents the total number of object descriptors present in the dictionary.

This \( N \) number of descriptors is again represented with the \( c \) number of class label i.e \( A = [A_1, A_2, \ldots, A_c] \) and each object class \( A_i = [a_{i1}, a_{i2}, a_{i3}, \ldots, a_{ik}] \) \( \in \mathbb{R}^{d^2 \times k_i} \) where \( k_i \) represents the number of training samples from class \( i \) and \( N = \sum_{i=1}^{c} k_i \) is the total number of object descriptors present in the dictionary. So an element \( a_{ij} \in \mathbb{R}^{d^2} \) from the dictionary \( A \) represents the \( j^{th} \) training sample of class \( i \).

For object classification, the true label of the test object is unknown. Thus \( y \) needs to be represented as a linear combination of all the training samples.

\[ y = Ax \] (6.2)
where \( x \in (x_{11}, x_{12}, \ldots x_{i,k_1}, \ldots, x_{ck_i}) \) is the coefficient vector corresponding to \( A \)

\[
y = [a_{i1}a_{i2}\ldots a_{ik}][x_{11}, x_{12}, \ldots x_{i3}, \ldots, x_{ck_i}]^T \tag{6.3}
\]

This equation can be approximate as

\[
y = [a_{i1}a_{i2}\ldots a_{ik}][x_{i1}, x_{i2}, \ldots x_{ik}]^T \tag{6.4}
\]

The above equation can be written as

\[
y \simeq A_ix_i \tag{6.5}
\]

An object can be classified into the \( i^{th} \) class if all or the maximum of the obtained nonzero coefficients of \( x \) are regrouped into the \( i^{th} \) class. Ideally, this can be stated as that the entries of \( x \) are all zero except those related to the same class as the test sample.

\[
y = Ax
\]

where \( x = [0, 0, \ldots, 0, x_{i1}, x_{i2}, \ldots, x_{im}, 0, \ldots, 0] \)

The residue \( r_i = \|y - A_i \times (\hat{x})_i\|_2 \) where \( i = 1, 2, \ldots c (\hat{x})_i \) denotes the entries of \( x \) associated with the training samples from the \( i^{th} \) class. Finally, \( x \) is assigned a class label \( i \) that resulted in the minimal residual so that sparse set solution of \( x \) can be obtained by solving the \( (\hat{x}) = \arg\min \| (\hat{x}) \| \).

Finding the solution to the above sparse problem is many. Several methods are there for solving this sparsity classification problem. We have used the Orthogonal matching pursuit method to find the sparse vector \( x \).

### 6.3.1 OMP Algorithm

Orthogonal matching pursuit(OMP) is one of the greedy based method for SR classification. This method is preferred for its speedy conversion rate. Here, in each iteration a local optimum solution is found. This is carried out by finding a column vector in \( A \) which is very close to the residual vector \( r \). Initially the residual vector is assigned to the test vector \( y \).
In each iteration, the residual vector is reassigned on the basis of evaluating $\arg\min_j |r_k - a_jx|^2$. In each iteration, one column vector is also selected on the basis of the minimum residual and then this selected column vector is removed for the next iteration. This process is repeated until the residual estimation reaches the maximum number of authorized iteration. As shown in the algorithm [6.1], OMP provide the solution to the approximation classification problem but it involve with the lots of computational task. This motivate us to develop the parallelization method for the OMP.

**Algorithm 6.1 OMP algorithm**

Input $y$ and $A$

output : $x^t$

Initialise $r_0 = y$, and $c_0 = \{\}$. 

for $k \geq 1$ to $t$ do 

Compute the residue $r_k = y - A\hat{x}$

Select a column $A_j$ such that $j^* = \arg\min_{j=1..N} |r_k - a_jx|^2$

Add $c_k = c_{k-1} \cup \{j^*\}$.

Coefficient Update : Update the coefficient of $(\hat{x}^k)$

if $||r_t||_2 < \epsilon$ or $t$ reaches the maximum number of authorized iteration then stop the algorithm.

end for

6.3.2 Dictionary Update

To increase the classification accuracy, each class is updated by constructing a new feature descriptor from the existing training sample and then add it to the dictionary. This process is repeated after the addition of $n$ number of test sample to the dictionary. The process of constructing new sample from the existing one is as follows. In the first step, each of the training sample $a_{ij}$ of class $A_i = [a_{i1} a_{i2} a_{i3} \ldots a_{ik}] \in R^{d \times k_i}$ are unfolded to form square matrix
of size $d \times d$ and then 1st frame choose as the referential frame and then next $(n - 1)$ frame are update with this referential one using the equation (4.29).

## 6.4 Parallel Implementation of OMP on GPU

Identifying the group of independent instruction from the program is the first step of parallelism. In OMP algorithm, inner product computation is a single operation which is executed for $n$ number of time for classifying the objects and is independent with respect to number of objects. Therefore, inner product has been parallelised using multiple thread.

### 6.4.1 GPU Execution Model

GPUs have been designed to run single instruction multiple data with the help of parallel thread. This processing unit generally consists of bundles of cards which consist of hundreds of processor cores. NVIDIA contribution CUDA provides the programmer the flexibility and ease of using the parallel programming. The function that executes the parallel computation on the GPU environment is called a kernel. A kernel is operated as a grid of thread blocks and all threads share data memory space. As show in the Figure 6.2, kernel is consists of one dimensional or the two dimensional grid. Each grid consists of group of thread where group of threads are executed concurrently. All thread in a block execute the same program.

Our parallel OMP algorithm 6.2 for the multi object tracking is described as follows: We have considered $p$ number of detected object for the classification. Input to our algorithm are $y^p$ and $A$ and the output is the coefficient vector $x_{tl}^t$. $y^p$ denote $p$ number of detected objects and $A \in R^{d \times N}$ is the dictionary that holds all the training samples. $x_{tl}^t$ represent the estimation of coefficient vector for object $l$ at $t^{th}$ iteration and $r_{tl}^t$ represent residual that is estimated for the object $l$ for the $t^{th}$ iteration. The process of classification run for $t^{th}$ number of times for classifying an object and in each iteration we need to find a column that give maximum similarity or the minimum residual
for the test object $y_l$. This column will be eliminated for the next iteration and its index is added to the sparse vector $x_l$. For different object different number of iteration are required for reaching the termination condition. Finally, after all the iteration are over for an object, then we classify the object on the basis of the retrieving the sparse coefficient of the vector $x_l$.

For implementing the above algorithm in NVIDIA’s GPU, we have launched the GPU kernel with two dimensional grid structure. As shown in the Figure 6.3, 2D grid structure provide adequate number of blocks for executing parallel thread. In our model, we have assigned each block to hold the dictionary along with a test object $y_l$ and run $N$ number of parallel thread. $N$ can be assigned with maximum value of 1024. Each block assigned with $N$ parallel thread is used to estimate the inner product of the test sample $y_l$ with the element of the dictionary. After $t^{th}$ number of iteration each block produce the coefficient vector $x_l$ for classifying the object $y_l$. 

Figure 6.2: cuda programming Model
Chapter 6. Parallelisation of Multi object Tracking On GPU

Figure 6.3: Execution of Thread for Computation of inner product

6.5 Performance Analysis

The proposed algorithm is implemented in CUDA, and runs on a laptop with an Intel Core (TM) i7 CPU and 4 GB RAM. The laptop is equipped with an NVIDIA GeForce GT 630M GPU card. This card has two multiprocessors and 96 CUDA cores with 1 GB of global memory. Proposed method is applied on the PET’s 09 dataset. We compared our proposed method with the existing methods of the literature ETHZ [61], EPFL [80] and PMTS [66].

Object tracking Evaluation: We applied our proposed parallel implementation on the PET’09 database. Visual result of this implementation is shown in Figure 6.4. It is noted that objects are detected for the longer sequence of 400 frames without any false detection. In Figure 6.5, we compared the detection of different objects in the frame for our proposed method with ETHZ [61], EPFL [80] and PMTS [66].

Performance Measurement Metrics: In addition to the visual comparison, multiple object tracking precision (MOTP) and multiple object tracking accuracy (MOTA) are two performance metrics [46] that are used
6.5 Performance Analysis

**Algorithm 6.2** Parallel OMP algorithm for Tracking

Input $y^p$ and $A$

output : $x^t_l$

Initialise $Q = 1, 2, ..., p, r^t_0 = y_t$, and $c_0 = \{\}$.

Select the Column $A_i$ that solve maximization problem.

for each $l \in Q$

$$c^t_l = c^t_{l-1} \cup \{\text{argmax}_{j=1..N} ||r^t_l - a_j x^t_l||_2\}$$

for each $t_l$ update residue

$$r^t_{l} = y_t - Ax^t_{l}$$

Coefficient Update : Update the coefficient of $(\hat{x}^k_{(t_l)})$

if $||r^t_{l}||_2 < \epsilon$ or $t_l$ reaches the maximum number of authorized iteration

then $Q = Q - l$, otherwise $t_l = t_l + 1$.

---

Figure 6.4: Visual tracking result

to compare our proposed method with the existing one. MOTP aims at evaluating the positive trajectories of the targets, while MOTA, composed of false negative rate, false positive rate and number of identity switches that measures the accuracy. We measured both MOTP and MOTA on the used benchmark. We obtained 72% of precision (MOTP) and 78% of accuracy (MOTA). These evaluation result shows the robustness and accuracy of our algorithm.

### 6.5.1 Execution Time

Execution time of the implemented OMP algorithm is evaluated and are compared for both GPU and CPU. Execution time of classification for one
Chapter 6. Parallelisation of Multi object Tracking On GPU

Figure 6.5: Visual comparison of our method with PMTS, ETHZ and EPFL

Table I: MOTP and MOTA evaluation results.

<table>
<thead>
<tr>
<th>$N$</th>
<th>MOTP</th>
<th>MOTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETHZ</td>
<td>54</td>
<td>52</td>
</tr>
<tr>
<td>EPFL</td>
<td>48</td>
<td>45</td>
</tr>
<tr>
<td>PMTS</td>
<td>68</td>
<td>72</td>
</tr>
<tr>
<td>Ours</td>
<td>72</td>
<td>78</td>
</tr>
</tbody>
</table>

person, two persons and five persons are estimated and shown in Table 6.1, Table 6.2 and Table 6.3 respectively.

Speed up rate of the GPU over the CPU are also shown in the Figure 6.6, From the figure it has been observed that GPU is computationally efficient as compared to the CPU and also processing time of GPU significantly reduced. We can note also that our approach performs better in terms of reducing the computation time when the number of objects increases more.
Table 6.1: Comparison of Processing time in CPU and GPU for 1 person

6.6 Conclusion

In this Chapter, we have presented a sparse representation based classification technique for multi object tracking algorithm for the detecting and tracking the objects. OMP algorithm is used for the classification of the object. objects are represented with covariance descriptor. Use of covariance descriptor increase the classification accuracy for different kind of objects. Furthermore, Our proposed algorithm is implemented with parallel thread on a graphical processing unit. This reduces considerably the processing time of algorithm for the object classification. Proposed algorithm shows the promising solution to ensure the detection of the moving object.
## Table 6.2: Comparison of Processing time in CPU and GPU for 2 person

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>GPU(ms)</th>
<th>CPU(ms)</th>
<th>Ratio(CPU/GPU)(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3</td>
<td>11</td>
<td>3.6</td>
</tr>
<tr>
<td>200</td>
<td>3.5</td>
<td>17</td>
<td>4.8</td>
</tr>
<tr>
<td>300</td>
<td>3.8</td>
<td>26</td>
<td>6.8</td>
</tr>
<tr>
<td>400</td>
<td>4</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>500</td>
<td>4.2</td>
<td>35</td>
<td>8.3</td>
</tr>
<tr>
<td>600</td>
<td>4.3</td>
<td>46</td>
<td>6.89</td>
</tr>
<tr>
<td>700</td>
<td>4.4</td>
<td>58</td>
<td>10.6</td>
</tr>
<tr>
<td>800</td>
<td>4.6</td>
<td>69</td>
<td>13.2</td>
</tr>
<tr>
<td>900</td>
<td>4.8</td>
<td>73</td>
<td>15</td>
</tr>
<tr>
<td>1000</td>
<td>5.1</td>
<td>82</td>
<td>16</td>
</tr>
</tbody>
</table>

## Table 6.3: Comparison of Processing time in CPU and GPU for 5 person

<table>
<thead>
<tr>
<th>Dictionary</th>
<th>GPU(ms)</th>
<th>CPU(ms)</th>
<th>Ratio(CPU/GPU)(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>4</td>
<td>25</td>
<td>6.25</td>
</tr>
<tr>
<td>200</td>
<td>4.5</td>
<td>40</td>
<td>8.8</td>
</tr>
<tr>
<td>300</td>
<td>4.9</td>
<td>48</td>
<td>9.7</td>
</tr>
<tr>
<td>400</td>
<td>5.1</td>
<td>54</td>
<td>10.5</td>
</tr>
<tr>
<td>500</td>
<td>5.3</td>
<td>65</td>
<td>12.2</td>
</tr>
<tr>
<td>600</td>
<td>5.6</td>
<td>88</td>
<td>15.7</td>
</tr>
<tr>
<td>700</td>
<td>7</td>
<td>112</td>
<td>16</td>
</tr>
<tr>
<td>800</td>
<td>7.4</td>
<td>135</td>
<td>18.2</td>
</tr>
<tr>
<td>900</td>
<td>7.2</td>
<td>140</td>
<td>19.4</td>
</tr>
<tr>
<td>1000</td>
<td>7.5</td>
<td>160</td>
<td>21.3</td>
</tr>
</tbody>
</table>
Figure 6.6: Speedup rate of GPU compared with CPU