Mean Shift Parallel Tracking on GPU

Peihua Li and Lijuan Xiao

School of Computer Science and Technology, Heilongjiang Univesity
Harbin, Heilongjiang Province, China, 150080
peihualj@hotmail.com

Abstract. We propose a parallel Mean Shift (MS) tracking algorithm on Graphics Processing Unit (GPU) using Compute Unified Device Architecture (CUDA). Traditional MS algorithm uses a large number of color histogram, say typically 16x16x16, which makes parallel implementation infeasible. We thus employ K-Means clustering to partition the object color space that enables us to represent color distribution with a quite small number of bins. Based on this compact histogram, all key components of the MS algorithm are mapped onto the GPU. The resultant parallel algorithm consist of six kernel functions, which involves primarily the parallel computation of the candidate histogram and calculation of the Mean Shift vector. Experiments on public available CAVIAR videos show that the proposed parallel tracking algorithm achieves large speedup and has comparable tracking performance, compared with the traditional serial MS tracking algorithm.

Keywords: Mean Shift tracking; Parallel algorithm; GPU; CUDA.

1 Introduction

Object tracking is an important research topic in the field of the computer vision. In recent years, the increasing demand for automated video analysis together with the advent of high quality, inexpensive video cameras and computers motivates more researchers to pay a great deal of attention to object tracking issues.

Mean Shift (MS) based tracking [1] is well known for its efficiency and performance. The MS tracking algorithm uses color histogram combined with spatial kernel to represent object distribution, and Bhattacharrya coefficient is used as similarity function that is optimized by Mean Shift iteration. Following [1], many researchers propose diverse algorithms for improving its performance. Collins solves the problem of kernel scale selection that exists in traditional MS tracking algorihtm [2]. In [3, 4], more rich spatial information is incorporated into object representation to increase robustness of the tracking algorithm. Tracking of objects with similarity or affine information are handled in [5, 6, 7]. Review of other MS tracking algorithms are omitted due to page limit.

A visual task of automated video analysis generally comprises object detection, object tracking, and recognition or encoding. A real time implementation of the visual task demands that each process should consume as little time as

H. Araujo et al. (Eds.): IbPRIA 2009, LNCS 5524, pp. 120–127, 2009.

possible. It is the most desirable that the intermediate process such as object tracking should be highly efficient. Hence, we develop a parallel MS tracking algorithm on GPU using CUDA that is implemented on GeForce 8800 GTS. The parallel algorithm solves the key issues concerned with computation of the candidate histogram and the mean shift vector. The resulting parallel MS consists of six kernel functions that corresponds to the principal components of its serial version.

Our paper is organized as follows. Section 2 reviews the serial MS tracking algorithm. The parallel tracking algorithm on GPU using CUDA is detailed in the section 3. In the section , The experiments are given in section 4 followed by the concluding remarks in section 4.

2 Serial Mean Shift Tracking Algorithm

In this section, we briefly review the traditional, serial MS tracking algorithm [1].

2.1 Target Model Representation

The target is represented by the rectangle region of width h_x and height h_y, centered on the origin. The target model is computed as

$$p_u = C \sum_{i=1}^{n} k\left(\frac{(x_i^*)^2}{h_x^2} + \frac{(y_i^*)^2}{h_y^2}\right)\delta_{ui}, \quad u = 0, \cdots, m-1 \tag{1}$$

where n is the number of pixels in the reference image, m is the number of bins, $k(\cdot)$ is the kernel function [1], and C is the normalization constant so that $\sum_{u=0}^{m-1} p_u = 1$.

2.2 Candidate Model Representation

The candidate model is defined by the following equation

$$q_u(\mathbf{z}) = C_h \sum_{i=1}^{n_h} k\left(\frac{(x-x_i)^2}{h_x^2} + \frac{(y-y_i)^2}{h_y^2}\right)\delta_{ui}, \quad u = 0, \cdots, m-1 \tag{2}$$

where the candidate image is centered on $\mathbf{z} = (x, y)$ that has n_h pixels with the spatial coordinates (x_i, y_i), C_h is the normalization constant and $\delta_{ui} = 1$ if the pixel with spatial coordidate $\mathbf{z}_i = (x_i, y_i)$ belongs to the uth bin, otherwise $\delta_{ui} = 0$.

2.3 Similarity Function (SF)

In our paper, we use Bhattacharyya coefficient as the similarity metric between the target model and the candidate model and it is defined by

$$S(\mathbf{p}, \mathbf{q}(\mathbf{z})) = \sum_{u=0}^{m-1} \sqrt{p_u q_u(\mathbf{z})} \tag{3}$$

2.4 Mean Shift Vector

The traditional MS tracking algorithm evaluates the object center position in a new frame by computing the MS vector $\varDelta \mathbf{z}$ iteratively until convergence or a prescribed maximum iteration number reaches. The formula for computing $\varDelta \mathbf{z}$ is as follows:

$$\varDelta \mathbf{z} = \frac{\sum_{i=1}^{n_h} (\mathbf{z}_i - \mathbf{z}) \omega_i g(\cdot)}{\sum_{i=1}^{n_h} \omega_i g(\cdot)} \tag{4}$$

where $g(\cdot) = k(\cdot)'$ and ω_i is given by $\omega_i = \sum_{u=0}^{m-1} \sqrt{\frac{p_u}{q_u(\mathbf{z}_k)}} \delta_{ui}$.

3 Parallel MS Tracking Algorithm on GPU

Compute Unified Device Architecture (CUDA) [8] is a new hardware and software architecture for managing computation on the GPU, which consists of extension of C Language for programming that makes it unnecessary to access graphics APIs. For the developers who are not acquainted with GPGPU, CUDA provides more flexibility and smooth transition from the serial algorithms to the parallel algorithms. In our experiments, we utilize the GPU card GeForce 8800 GTS manufactured by NVIDIA's corporation.

3.1 Color Space Partition

The color space in the traditional MS tracking algorithm is uniformly partitioned into m bins, typically 16x16x16, which makes parallel algorithm infeasible. Noticing that object color is usually compact and distributed only in some small regions in the whole color space, as done in [9], we use K-Means clustering method to partition the color space and then construct lookup table.

3.2 Parallel Tracking Algorithm with CUDA

Before invoking kernel functions, we first compute the bin indices of candidate pixels according to lookup table with CPU. In order to avoid frequent data transfer between CPU memory and GPU memory, we compute the bin indices of a sub-image that is centered at the object center and that is double size of the object. Then we bind (copy) the 2D index array to the texture memory on GPU. The advantage of texture memory is that its access is optimized for 2D spatial locality by GPU.

Kernel #1. The purpose of the kernel is to calculate the n sub-histograms and n stands for the number of blocks. Each thread loads one bin index u from the texture memory. During computation of the candidate histogram, when different threads simultaneously write data into the same shared memory address, memory collision is inevitable. In order to solve the problem, we follow the tactic that one thread maintains its own sub-histogram $[q'_{(i,0)}(\mathbf{z}), q'_{(i,1)}(\mathbf{z}), \cdots, q'_{(i,m-1)}(\mathbf{z})]$,

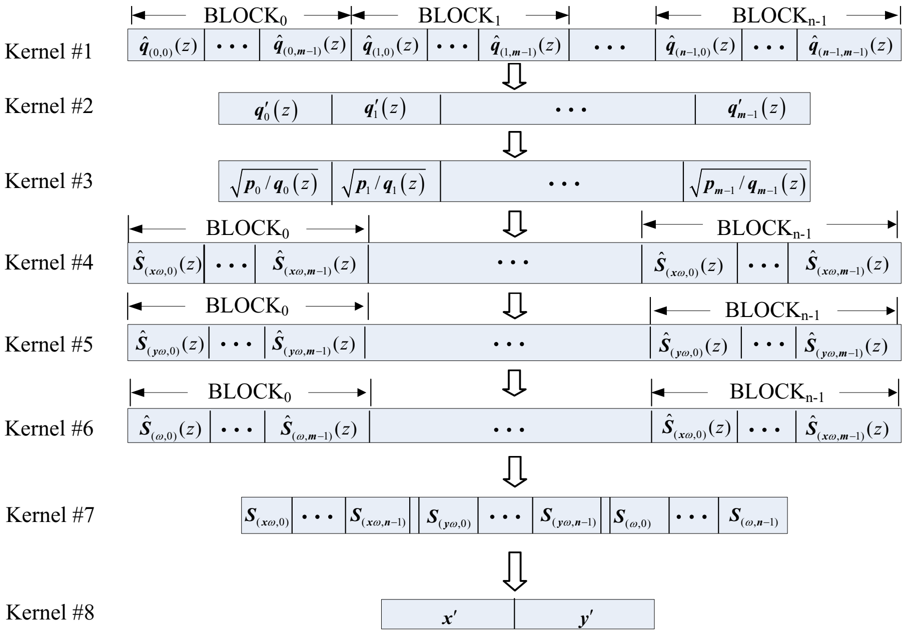


Fig. 1. Six kernels results in global memory

$i = 0, \cdots, l-1$, where l is the number of threads in a block. At this time, all the computed sub-histograms are written into the shared memory. The advantage of this method is that each thread writes data into different shared memory addresses and there is no memory collision. Then we merge the sub-histograms within block. For the jth block, $j = 0, 1, \cdots, n-1$, we have $\hat{q}_{(j,u)}(\mathbf{z}) = \sum_{i=0}^{l-1} q'_{(i,u)}(\mathbf{z}), (u = 0, 1, \cdots, m-1)$. And then the n sub-histograms $[\hat{q}_{(j,0)}(\mathbf{z}), \hat{q}_{(j,1)}(\mathbf{z}), \cdots, \hat{q}_{(j,m-1)}(\mathbf{z})]$ are written into the global memory, which are seen in the Fig. 1.

Remark. The size of the shared memory in CUDA is limited by $16KB$, which is a big challenge for the developers. In our experiments, we empirically set the number of bins to 13. So, noticing that the data type in histogram is single-precision floating number, if the number of threads per block is set to 256, we can only have one active block per multiprocessor ($\lfloor(16*1024)/(13*256*4)\rfloor = 1$), which greatly impacts the parallelism of the tracking algorithm.

Kernel #2. The kernel aims at merging the n sub-histograms produced by the kernel #1, into the complete m candidate histograms without being normalized (CHWN) $\mathbf{q}'(\mathbf{z}) = [q'_0(\mathbf{z}), q'_1(\mathbf{z}), \cdots, q'_{m-1}(\mathbf{z})]$. Load the n sub-histograms into the shared memory and for the jth block, the uth CHWN is produced by $q'_u(\mathbf{z}) = \sum_{j=0}^{n-1} q'_{(j,u)}(\mathbf{z})$, using the parallel sum reduction algorithm, which is introduced below. But the algorithm demands that the data must be a power of two. To handle non-power-of-two data, while loading the sub-histograms from the global memory, the kernel pads the shared memory array and initializes the extra memory to zero.

Kernel #3. First load the m results from the kernel #2 into the shared memory. To obtain the candidate histogram $\mathbf{q}(\mathbf{z}) = [q_0(\mathbf{z}), q_1(\mathbf{z}), \cdots, q_{m-1}(\mathbf{z})]$, we compute the $C_h = \frac{1}{\sum_{u=0}^{m-1} q'_u(\mathbf{z})}$ using the parallel sum reduction algorithm and then caculate $\mathbf{q}(\mathbf{z}) = [C_h q'_0(\mathbf{z}), C_h q'_1(\mathbf{z}), \cdots, C_h q'_{m-1}(\mathbf{z})]$, and $\sqrt{\frac{p_u}{q_u(\mathbf{z})}}$, $(u = 0, 1, \cdots, m-1)$.

Kernel #4 #5 #6 #7. To calculate the Mean Shift vector $\Delta\mathbf{z} = \frac{\sum_{i=1}^{n_h}(\mathbf{z}_i - \mathbf{z})\omega_i}{\sum_{i=1}^{n_h}\omega_i}$, we must first compute $\sum_{i=1}^{n_h} x_i\omega_i$, $\sum_{i=1}^{n_h} y_i\omega_i$ and $\sum_{i=1}^{n_h}\omega_i$, where we use Epanechnikov profile and $g(\cdot)$ is constant. Noticing $\omega_i = \sqrt{\frac{p_u}{q_u(\mathbf{z}_k)}}\delta_{ui}$, the pattern of the above three summations is similar to the pattern of CHWN $q'_u(\mathbf{z}) = \sum_{i=1}^{n_h} k(\cdot)\delta_{ui}$. Hence, in a similar way, Kernel #4, #5 and #6 respectively produce the n sub-sums of m entries for the three summations. Kernel #7 merges the n sub-sums to get $[S_{(x\omega,0)}, \cdots, S_{(x\omega,n-1)}]$, $[S_{(y\omega,0)}, \cdots, S_{(y\omega,n-1)}]$, and $[S_{\omega_0}, \cdots, S_{\omega_{n-1}}]$.

Kernel #8. Using the parallel sum reduction algorithm, we can compute the new object center $\mathbf{z}' = (x', y')$, where $x' = \frac{\sum_{j=0}^{n-1} S_{(x\omega,j)}}{\sum_{j=0}^{n-1} S_{\omega_j}}$ and $y' = \frac{\sum_{j=0}^{n-1} S_{(y\omega,j)}}{\sum_{j=0}^{n-1} S_{\omega_j}}$.

The new center $\mathbf{z}' = (x', y')$ is transferred from GPU to CPU and the latter can determine whether iteration process should continue, by verifying whether $\|\mathbf{z} - \mathbf{z}'\| < \varepsilon_{\mathbf{z}}$, or maximum iteration number N reaches.

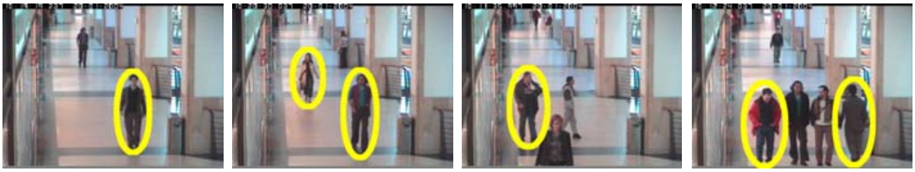
4 Experiments

The proposed parallel tracking algorithm is implemented on GPU card GeForce 8800 GTS, which is installed on a PC with 3.0GHz Intel Pentium(R) 4 CPU and 2G Memory. In all experiments, the 16x16x16 color histogram is used in the serial MS algorithm and the cluster number K is set to 13 in the parallel algorithm.

4.1 Data Set and Performance Comparision

We use publicly available benchmark data sets of CAVIAR project [10], where ground truth has been labeled. Two video clips of OneStopEnter2cor.mpg (OSE) and ThreePastShop1cor.mpg (TPS) are used in which six objects are tracked. Please see Fig. 2 for reference.

In Table 1, the 4th and 5th columns give average distance error and average overlapping region error [11] in the format of mean plus/minus standard variance (mean ± std), which shows that the two algorithm are comparable in performance. The 6th column lists the average tracking time of CPU and GPU respectively, and the speedups are provided in the last column of Table 1.



(a) ID3 in OSE (b) ID4 and ID6 in OSE (c) ID0 in TPS (d) ID1 and ID4 in TPS

Fig. 2. Six objects that are tracked in our experiments

Table 1. Comparison between GPU and CPU

Scenarios	Object ID	Algorithm	Distance error (pixel)*	Overlapping region error(%)*	Tracking time (ms)	SpeedUp
OSE	ID3	CPU	18.47±11.23	0.361±0.221	5.69	2.44
		GPU	12.28±06.38	0.345±0.210	2.33	
	ID4	CPU	12.68±04.63	0.446±0.221	5.49	2.52
		GPU	15.21±07.12	0.478±0.185	2.18	
	ID6	CPU	07.10±03.61	0.351±0.179	6.38	2.44
		GPU	8.58±02.49	0.456±0.153	2.61	
TPS	ID0	CPU	15.43±08.20	0.246±0.165	8.98	2.79
		GPU	15.05±06.75	0.232±0.164	3.22	
	ID1	CPU	12.96±13.29	0.182±0.117	9.01	3.36
		GPU	07.21±06.41	0.183±0.114	2.68	
	ID4	CPU	08.00±06.25	0.266±0.100	5.88	2.77
		GPU	06.57±02.53	0.362±0.165	2.12	

* The format of data is mean±std.

4.2 Effect of Bandwidth between CPU and GPU on the Parallel Algorithm

It is well-known that data transfer between CPU and GPU is the bottleneck for many GPGPU applications. In our algorithm, we first need to bind (copy) to the texture the bin index array of three times the size of the object. Take ID4 in TPS sequence as an example, this operation takes 0.406ms. During mean shift iterations, we have to transfer the object position $\mathbf{z}' = (x', y')$ from GPU to CPU so that the latter can determine whether to halt the iteration, which takes 0.0188∗5=0.094ms, assuming that average iterative number is 5. Finally, to compute the $S(\mathbf{p}, \mathbf{q}(\mathbf{z}))$, we must copy the candidate histogram from GPU to CPU as well that takes 0.0181ms. To sum up, data transfer between GPU and CPU takes 0.518ms. And the speedup will reach 3.67 if the data transfer cost is neglected.

5 Conclusions

The paper proposes a parallel MS tracking algorithm on GPU using CUDA. Using K-Means clustering to partition the object color space, we can represent object distribution with quite a very small number of bins which makes parallel MS algorithm possible. Then we map the major components in serial MS tracking algorithm onto the GPU, which involves primarily computation of the candidate histogram and the calculation of the MS vector. Experimental results show that the proposed parallel algorithm achieves large speedups. Meanwhile, it has comparable tracking performance with the traditional serial MS tracking method.

In the kernel functions (#1, #4, #5, #6), the number of active blocks per multiprocessor is severely limited by the $16KB$ shared memory that results in low parallelism of the algorithm. It is our concern to think about effective technique to increase the parallelism. In our experiments the number of histogram bins is empirically chosen. It is necessary in future work to determine adaptively the number of bins for better performance of the parallel algorithm.

Acknowledgment

The work was supported by the National Natural Science Foundation of China under Grant 60673110 and Natural Science Foundation of Heilongjiang Province (F200512), supported in part by Program for New Century Excellent Talents of Heilongjiang Province (1153-NCET-002), Sci. & Tech. Research Project of Educational Bureau of Heilongjiang Province (1151G033), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry and Ministry of Personnel of China, Sci. and Tech. Innovation Research Project (2006RFLXG030) of Harbin Sci. & Tech. Bureau.

References

1. Comaniciu, D., Ramesh, V., Meer, P.: Real-time Tracking of Non-rigid Objects Using Mean Shift. In: Proc. IEEE Conf. Comp. Vis. Patt. Recog., pp. 142–149. Hilton Head Island, South Carolina (2000)
2. Collins, R.T.: Mean-shift Blob Tracking Through Scale Space. In: Proc. IEEE Conf. Comp. Vis. Patt. Recog., Madison, Wisconsin, pp. 234–241 (2003)
3. Birchfield, S.T., Rangarajan, S.: Spatiograms versus Histograms for Region-based Tracking. In: Proc. IEEE Conf. Comp. Vis. Patt. Recog., San Diego, CA, USA, pp. 1158–1163 (2005)
4. Zhao, Q., Tao, H.: Object Tracking using Color Correlogram. In: IEEE Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS) in conjunction with ICCV, pp. 263–270 (2005)
5. Yilmaz, A.: Object Tracking by Asymmetric Kernel Mean Shift with Automatic Scale and Orientation Selection. In: Proc. IEEE Conf. Comp. Vis. Patt. Recog., Minneapolis, Minnesota, pp. 1–6 (2007)

6. Zhang, H., Huang, W., Huang, Z., Li, L.: Affine object tracking with kernel-based spatial-color representation. In: Proc. IEEE Conf. Comp. Vis. Patt. Recog., San Diego, CA, USA, pp. 293–300 (2005)
7. Leichter, I., Lindenbaum, M., Rivlin, E.: Visual Tracking by Affine Kernel Fitting Using Color and Object Boundary. In: Proc. Int. Conf. Comp. Vis., Rio de, Janeiro, Brazil, pp. 1–6 (2007)
8. NVIDIA CUDA Homepage, `http://developer.nvidia.com/object/cuda.html`
9. Li, P.: A clustering-based color model and integral images for fast object tracking. Signal Processing: Image Communication, 676–687 (2006)
10. EC funded CAVIAR project/IST 2001 37540 (2004),
`http://homepages.inf.ed.ac.uk/rbf/CAVIAR/`
11. Bajramovic, F., Grabl, C., Denzler, J.: Efficient Combination of Histograms for Real-Time Tracking Using Mean-Shift and Trust-Region Optimization. In: Proc. 27th DAGM Symposium on Pattern Recognition, Vienna, Austria, pp. 254–261 (2005)