PARALLEL VIDEO PROCESSING USING EMBEDDED COMPUTERS

Bo Fu[1], Anup Mohan[2], Yifan Li[1],
Sanghyun Cho[1], Kent Gauen[1], and Yung-Hsiang Lu[1]
[1]Purdue University, West Lafayette, IN, USA, [2]Intel Corporation, Santa Clara, CA

***Abstract*—As frame rates and resolutions of video streams increase, a need for parallel video processing emerges. Most studies offload computation to the cloud, but this is not always possible. For example, solar-powered cameras can be deployed in locations away from power grids. A better choice is to process the data locally on embedded computers without raw video transmission through networks. Parallel computing alleviates the performance bottleneck of a single embedded computer but it degrades analysis accuracy because partitioning video streams breaks the continuity of motion. This paper presents a solution for maintaining accuracy in parallel video processing. A video stream is divided into multiple segments processed on different embedded computers. The segments overlap so that continuous motion can be detected. The system balances workload based on the speed of GPU and CPU to reduce execution time. Experimental results show up to 82.6% improvement in accuracy and 58% reduction in execution time.**

Keywords*—*Video Processing, Parallel Processing, Embedded Computers, Workload Balance.

I. INTRODUCTION

Video surveillance has been widely used for many purposes, such as monitoring restricted areas, traffic congestion, and building entrances. To run these surveillance applications, network cameras may record videos and send the videos to the cloud for processing. Performing computation in the cloud has two drawbacks. First, the virtual machines from cloud providers can be geographically far away from the network cameras. As this distance increases, the frame rate drops [1][2]. Dropping frames affects the accuracy of video processing such as losing track of objects. Second, solar-powered network cameras can be deployed in locations away from power grids. This restriction requires network cameras to compute locally and send only the result. A natural choice to meet this requirement is to use embedded computers as the computing engine and place them close to cameras.

Processing video streams can be computationally expensive, usually beyond the capability of a single embedded computer. There is a need for using multiple computers to share the computation. Although parallel processing has been studied in many aspects (such as Hadoop-based video processing and parallel processing using GPU [5]-[9]), there are still unsolved problems. First, partitioning video breaks the continuity of motion and may lead to incorrect analysis results. Second, few studies consider how to utilize the capacity of both CPU and GPU to reduce execution time. Third, although platforms such as Hadoop use speculative execution that repositions slow tasks to other computers to achieve workload balance, it takes extra time to detect the slow tasks and re-execute them. There is a need for a method to manage resource efficiently and balance workload as early as possible to avoid re-execution.

This paper presents a solution for parallel video processing, with the emphasis on counting moving objects. The method divides the video into multiple segments; two adjacent segments share multiple frames (referred to as an overlap section) so that the objects appearing in more than one segment will not be counted repeatedly. Objects are detected and tracked in the overlap section. These objects are removed from the counting results because they have been counted in the previous segment. This method can avoid objects being counted twice (referred as double counting) due to their appearance in two video segments. The system also decides the number of frames to be processed by each computing engine (including CPU and GPU) for each embedded computer. It performs test runs on all the embedded computers that process several video frames before processing the whole video segment and records the execution time for the frames. The number of frames for each computing engine is decided through the information recorded in the test run. This study uses a people counting program [3] as the target application to evaluate the system. The experimental results show up to 58% reduction in the execution time by selecting computing engine for each embedded computer (referred as resource allocation later). The overlapping method eliminates the double counting rate by 82.6%. The major contributions of this paper include:

1) A parallel video processing system performing computation on multiple embedded computers.
2) A method to reduce double counting.
3) A method that decides the number of frames each computing engine should process to reduce execution time.

TABLE I: Comparison of different studies. **DCE**: double counting elimination.

Publication	Parallel	Resource	DCE	Computer
Wang et al. [4]	no	CPU	–	PC
Ryu et al. [5]	yes	CPU	no	PC
Tan et al. [6]	yes	CPU	no	PC
Zhao et al. [7]	yes	CPU	no	PC
Zhang et al. [8]	yes	CPU	no	PC
Aradhye et al. [9]	yes	GPU	no	PC
This paper	yes	both	yes	Embedded

II. RELATED WORK

Researchers have addressed the issue of parallel video analytics. Ryu et al. [5] propose an extensible video processing framework in Hadoop; video analytics is processed in map tasks. Tan et al. [6] suggest another framework that uses map tasks to group frames, then executes the video analytics with reducing tasks. Zhao et al. [7] build HPVI that extends Hadoop

to support video analytic applications. HVPI provides a video read-and-write interface for users to quickly create large-scale video analytic applications based on Hadoop and allows users to port their existing video analytic applications written in C/C++ into Hadoop platform. Zhang et al. [8] propose a framework that uses Storm[10] for object detection in real-time. Aradhye et al. [9] reduce the latencies in video analysis process by porting its compute intensive parts to a GPU cluster.

Object tracking algorithms are the basis of video surveillance applications. Zhang et al. [11] propose a network flow based optimization method for data association to track multiple objects. The maximum-a-posteriori data association problem is mapped into a cost-flow network with a non-overlap constraint on trajectories. Leibe et al. [12] consider object detection and spacetime trajectory estimation as a coupled optimization problem. It is formulated in a hypothesis selection framework and builds upon a pedestrian detector. Zhang et al. [3] propose a tracking-by-detection framework for multi-person tracking. The appearance model is formulated as a template ensemble updated online given detections provided by a pedestrian detector. They employ a hierarchy of trackers to select the most efficient tracking strategy and an algorithm based on a temporal sliding window to adapt the conditions for trackers' initialization and termination.

As shown in Table I, this paper is different from other studies in the following aspects: (1) The parallel video analytics framework is built on embedded computers. (2) The system preserves continuity in partitioned video segments so that double counting will be reduced. (3) The system balances the workload of computing engines to reduce execution time.

III. System Design

This section describes the system for processing video on embedded computers. The system contains two major components: the manager and the workers. The manager first checks the speed of workers through test runs and decides the number of frames each worker should process (described in Section III-A). The manager then partitions the video into different segments with some shared frames to eliminate double counting (described in Section III-B). Afterward, it generates a job-specific ID to identify a job, and then sends the video segments and the ID to the workers. A worker uses either a CPU or a GPU to process the video segment (described in Section III-C). After the worker finishes the processing, it sends the results and a message with the ID to the manager. The manager checks if processing is finished in all workers and merges the results.

A. *Balanced Video Allocation*

The speed of each type of computing engine (CPU or GPU) may vary for different video streams. In Figure 1, it takes an average of 400 and 124 milliseconds to process a frame using GPU and CPU respectively, indicating the CPU is around 3 times faster than GPU. The reason is that too few GPU cores, which are much slower than CPU cores, are used for processing low-resolution frames. As the resolution becomes larger, the processing speed of GPU exceeds that of CPU since more GPU cores are used. Thus the manager should give more frames to the computing engine with higher speed to balance the workload among different computing engines. To do this, the manager tests the speed of computing engines

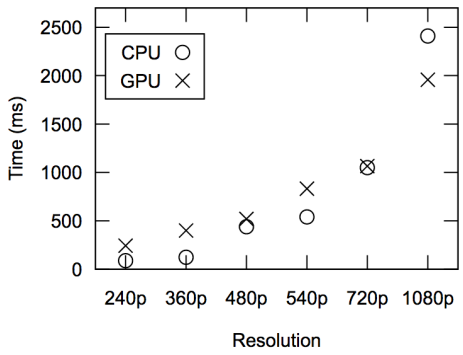


Fig. 1: The average execution time per frame of CPU and GPU with different resolutions. The results are calculated from performing people counting on the first 20 frames using TK1[13]. **240P**: 252×240. **360p**: 480×360. **480p**: 858×480. **540p**: 720×540. **720p**: 1280×720. **1080p**: 1920×1080.

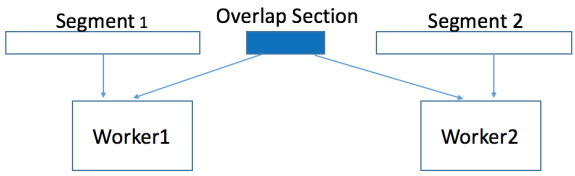


Fig. 2: The overlap section (blue box) between two consecutive segments. The overlap section is sent to both workers.

for different videos by sending several frames to each worker for the test run. During the test run, the workers process the frames and record the processing time using CPU and GPU respectively. Each worker calculates the average number of frames processed in one second and submits this value to the manager.

The manager collects the speed of computing engines from all the workers. The number of frames that should be given to a certain computing engine is as a proportion to the speed of this computing engine. For example, suppose there are two computers. The CPU and the GPU of computer 1 (denoted as C-1 and G-1) can process 10 frames and 15 frames each second respectively. The CPU and the GPU of computer 2 (denoted as C-2 and G-2) can process 5 frames and 20 frames per second. If there are 50 frames in total, the number of frames assigned to C-1, G-1, C-2 and G-2 should be 10, 15, 5, and 20 (in proportion to the speed). This method has two advantages. First, unlike other studies that consider either CPU or GPU, this method uses both. Second, this method balances the workload among all workers and thus reduces execution time.

B. *Double-counting Elimination*

To eliminate double counting, two consecutive segments share some frames (called overlap section, as shown in Figure 2). The overlap section helps determine the number of objects already counted in the previous video segment. More specifically, before the worker counts objects in a video segment, it first detects and counts the objects (referred as DC-Objects) in the overlap section. These DC-Objects are regarded to have been already counted in the previous video segment since the counting of these objects happens in the

overlap section shared by both video segments. When the manager merges the results from all workers, it excludes what is regarded as double-counted objects by subtracting the number of DC-Objects detected in the overlap section. The revised counting result C (after double counting elimination) is calculated as:

$$C = \sum_{i=1}^{n-1} (S_i - O_i) \quad (1)$$

where S_i and O_i stand for the number of objects counted in the video segment and the number of people counted in the overlap section in the ith video segment respectively.

As a corner case, if a walking person stops in the overlap section and starts moving again after the overlap section, the person will be counted twice but not be eliminated during overlap section. This is because this person is regarded as the background in the overlap section if the person does not move. This paper currently does not handle this case. In the future, we can find stationary people in overlap sections by matching features using SURF[14] or SIFT[15] algorithms.

C. *People Counting Program*

The people counting program used by the workers for processing is built based on the multi-object people tracking algorithm proposed by Zhang et al. [3]. This method counts the people walking into the camera scene. More specifically, a person is counted when a new tracker with a unique ID is created to track this person. As the person is being tracked, the person will not be counted again. Object detection is the most computationally intensive part, and can be executed on either CPU or GPU, which is controlled by the video processing system.

IV. EVALUATION

A. *Evaluation Setup*

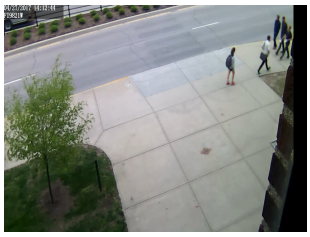
Hardware: The manager runs on a Rasberry Pi model 3 and the workers run on three Jetson Tegra K1[13] (referred as TK1). The manager keeps all cores in TK1 on for better performance. All computers are interconnected through a NETGEAR router and can communicate with each other using local IP addresses.

Software: The system is implemented in C++ with OpenCV-2 and uses Rsync for network communication over embedded computers. The test run is written in bash scripts.

Input Video: This study uses videos with the resolution of 1280×720 (referred as large video) and 640×230 (referred as small video) respectively. The two videos are both campus street scene recorded from the FOSCAM network camera. Figure 3a, Figure 3b and Figure 3c are snapshots from the two videos. To maintain the detection precision, the small video is clipped from video 2 without losing pixel intensity.

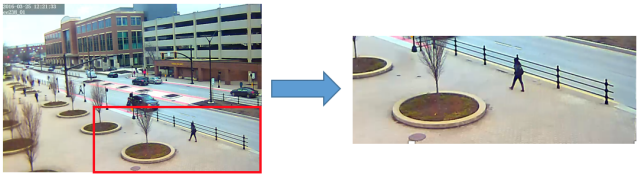
B. *Execution Time*

The performance is evaluated by comparing execution time among three resource allocation strategies: performing computation only on CPUs (used by [5]-[8]), performing computation only on GPUs (used by [9]) and performing computation on both CPUs and GPUs (this paper). Table II shows for both large and small video, this paper achieves the lowest execution time. We also observe performing computation on CPU for the small video takes 80 seconds. This is much shorter compared



(a) Video 1.

(b) Video 2.



(c) Small video.

Fig. 3: (a), (b): snapshots of the large videos with people walking along the sidewalk. (c) a small video is clipped from the the bottom right (red rectangle) of video 2.

TABLE II: Comparison of the execution time of different resource allocation strategies. S-C and S-G: processing small video using only CPU and only GPU. L-C and L-U: processing large video using only CPU and only GPU. Mix: the strategy used in our system, with utilization of both CPU and GPU.

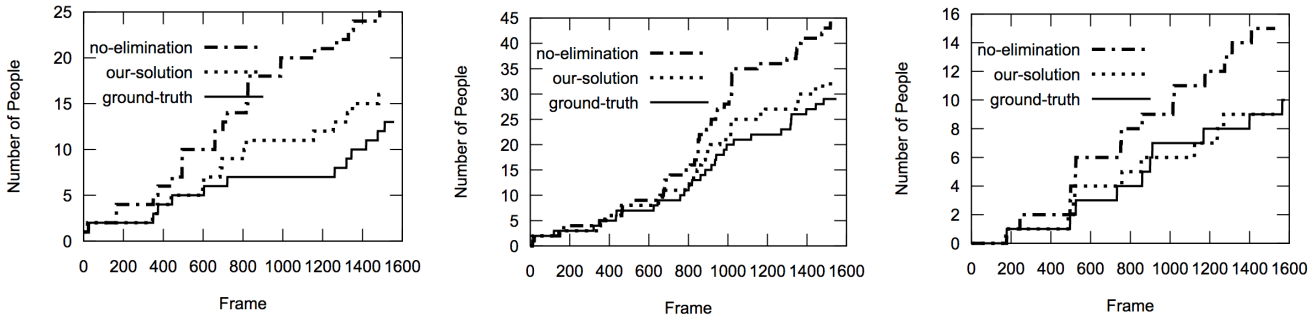
Condition	S-C	S-G	S-Mix	L-C	L-G	L-Mix
Time (s)	80	152	65	1029	1001	435

to the 152 seconds using GPU. This further demonstrates the processing speed of the CPU is superior to the GPU for the frames with small data with few computations, as described in Section III-A.

C. *Accuracy*

The analysis of accuracy is done using a video of a single location from different times of day: the morning, noon, and afternoon. Figure 4 shows the number of people counted as the frame number increases. Processing videos sequentially (the solid line) without partitioning is the ground truth, since no double counting happens. The dotted lines in the figures indicate the parallel counting solution of this paper. The dashed lines represent the parallel processing without eliminating double counting (used by [5]-[9]).

As shown in Figure 4, the dotted lines are closer to the solid lines compared with the dashed lines for all video clips, indicating the counting result is more accuracy by eliminating double counting. Figure 4(a) and Figure 4(b) both show the dotted lines higher than the solid lines, indicating the system still does not remove some double-counted people after elimination. In Figure 4(c), the dotted line intersects with the solid line, indicating the elimination sometimes removes more people than actually double-counted. The two observations occur because the counting program uses background subtraction, which is adaptive and varies slightly through time, to detect objects. As a result, counting the same overlap section of two consecutive video segments may give different result because they have the different adaptive backgrounds.



(a) morning (b) noon (c) afternoon

Fig. 4: Results for three videos. (a) is recorded at 2016-03-29 19:21:44 EST. (b) is recorded at 2016-03-25 12:21:33 EST. (c) is recorded at 2016-03-25 16:26:57 EST. The location is the EE builing, West Lafayette, Indiana, U.S.A. (40.428643, -86.911919). The videos have the frame rate of 30 frames per second.

It is also observed that the counting result in (b) and (c) are more accurate than (a). The reason could be the direction of the sunlight. Due to the camera angles, the video captured in the morning is brighter than the other two, and the shadows of the people are longer. This decreases the program's ability to eliminate double-counted people in the overlap sections.

The accuracy is as defined:

$$Accuracy = 1 - \frac{|R - R_t|}{R} \tag{2}$$

where R stands for the experimental result and R_t represents the ground truth (the solid line). The experimental results show that this paper achieves an average of 89.7% accuracy and eliminates 82.6% of double-counted people for the three videos.

D. *Workload Balance*

We evaluate workload balance by recording the execution time of all the computing engines. We compare the balanced allocation (considering workload balance) to equal-sized video allocation (used by [5]-[8]). Table III shows the number of frames processed by each computing engine and their execution time. It is observed that the processing time of S-1 lies between 37 second and 45 seconds (18% difference), while the time of S-2 varies from 37 seconds to 59 seconds (37% difference). For large video, L-1 and L-2 show 14% and 51% difference respectively. For both the large and the small video, this paper has much smaller difference in execution time for all engines, which indicates less idle time for fast engines and thus better balance. Table III also displays the Standard Deviation (SD) of the processing time for all computing engines. The result shows the standard deviations are 3.6 and 17.6 for the smaller video, much smaller than 7.5 and 81.3 from equal-sized allocation method.

E. *The Lengths of Overlap Sections*

Table IV shows the relationship between the length of the overlap sections and accuracy. As the length increases, the execution time becomes larger and the accuracy improves. It is also shown that for both the large and the small video, the accuracy saturates when the length of overlap section is 5.

TABLE III: Processing time and the number of frames processed by each computing engine. C-1 to C-3: the CPU of the three TK1 computers. G-1 to G-3: the GPU of the three computers. S-1 and L-1: balanced allocation using small and large video. S-2 and L-2: equal-sized allocation for small and large video. **SD**: Standard Deviation.

	Engine	C-1	G-1	C-2	G-2	C-3	G-3	SD
S-1	**Frames**	330	210	250	250	310	230	-
	Time(s)	45	41	44	37	47	45	3.6
S-2	**Frames**	263	263	263	263	264	264	-
	Time(s)	37	41	46	46	48	59	7.5
L-1	**Frames**	220	360	280	350	260	330	-
	Time(s)	320	346	343	330	317	299	17.6
L-2	**Frames**	300	300	300	300	300	300	-
	Time(s)	363	324	459	273	364	225	81.3

TABLE IV: The relationship between the length of overlapped frames and accuracy.

	Overlap Length	0	1	3	5	7
Small	**Accuracy(%)**	64.4	64.4	82.9	87.9	87.9
	Time(s)	63	63	64	65	66
Large	**Accuracy(%)**	68.3	68.3	91.7	91.7	91.7
	Time(s)	428	430	430	435	438

V. CONCLUSION

This paper presents a solution for parallel video processing on embedded computers. The solution divides video streams into multiple segments. Each two consecutive segments overlap to detect continuous motion. The system balances the workload for workers by collecting the speed of each worker in advance. The example of people counting shows that the solution achieves up to 89.7% accuracy and eliminates 82.6% of the double-counting cases. The resource manager achieves up to 58% speedup. The results may provide insight for building future network cameras that will release the need for heavy video data transmission over the long-distance network.

ACKNOWLEDGEMENTS

This project is supported in part by National Science Foundation ACI-1535108. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] W. Chen, A. Mohan, Y.-H. Lu, T. Hacker, W. T. Ooi, and E. J. Delp, "Analysis of large-scale distributed cameras using the cloud," *IEEE Cloud Computing*, vol. 2, no. 5, pp. 54–62, 2015.

[2] A. Mohan, A. S. Kaseb, Y.-H. Lu, and T. J. Hacker, "Location based cloud resource management for analyzing real-time videos from globally distributed network cameras," in *Cloud Computing Technology and Science (CloudCom)*, pp. 176–183, IEEE, 2016.

[3] J. Zhang, L. L. Presti, and S. Sclaroff, "Online multi-person tracking by tracker hierarchy," in *IEEE International Conference on Advanced Video and Signal-Based Surveillance*, pp. 379–385, 2012.

[4] Y. Wang, W.-T. Chen, H. Wu, A. Kokaram, and J. Schaeffer, "A cloud-based large-scale distributed video analysis system," in *IEEE International Conference on Image Processing*, pp. 1499–1503, 2016.

[5] C. Ryu, D. Lee, M. Jang, C. Kim, and E. Seo, "Extensible video processing framework in apache hadoop," in *IEEE International Conference on Cloud Computing Technology and Science*, vol. 2, pp. 305–310, 2013.

[6] H. Tan and L. Chen, "An approach for fast and parallel video processing on apache hadoop clusters," in *IEEE International Conference on Multimedia and Expo*, pp. 1–6, 2014.

[7] X. Zhao, H. Ma, H. Zhang, Y. Tang, and Y. Kou, "Hvpi: extending hadoop to support video analytic applications," in *IEEE International Conference on Cloud Computing*, pp. 789–796, 2015.

[8] W. Zhang, P. Duan, Q. Lu, and X. Liu, "A realtime framework for video object detection with storm," in *IEEE International Conference on Ubiquitous Intelligence and Computing*, pp. 732–737, 2014.

[9] T. Abdullah, A. Anjum, M. F. Tariq, Y. Baltaci, and N. Antonopoulos, "Traffic monitoring using video analytics in clouds," in *IEEE/ACM International Conference on Utility and Cloud Computing*, pp. 39–48, 2014.

[10] "Storm documentation." http://www.storm-project.net/.

[11] L. Zhang, Y. Li, and R. Nevatia, "Global data association for multi-object tracking using network flows," in *IEEE International Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[12] B. Leibe, K. Schindler, and L. Van Gool, "Coupled detection and trajectory estimation for multi-object tracking," in *IEEE International Conference on Computer Vision*, pp. 1–8, 2007.

[13] "Jetson tk1 embedded development kit." http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html.

[14] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," *ECCV*, pp. 404–417, 2006.

[15] D. G. Lowe, "Object recognition from local scale-invariant features," in *International Conference on Computer Vision.*, vol. 2, pp. 1150–1157, Ieee, 1999.