Parallel Background Subtraction with KNN

吳雅就
National Chiao Tung University
EC-617
yacwu@cs.nctu.edu.tw

顏義洋
National Chiao Tung University
EC-617
scps950707@gmail.com

巫謹佑
National Chiao Tung University
EC-617
jinyo840121@gmail.com

ABSTRACT

In this project, we use a variety of parallel frameworks to parallelize a commonly used algorithm of background subtraction in OpenCV, include OpenMP, Pthreads, OpenCL, and CUDA. And after finishing the project, we use our experience of this project to rewrite some codes and contribute them to the OpenCV repository. Parts of them are already merged to the master branch and will appear in the next release. See Pull request #10432 and Pull request #10553.

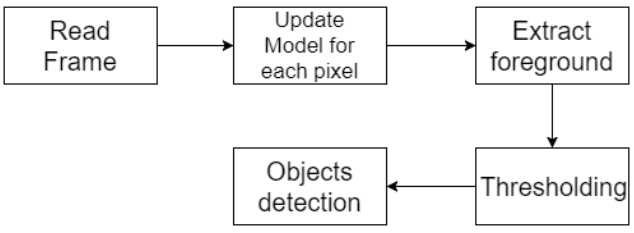
1. INTRODUCTION

In computer vision, background subtraction is a commonly used approach for detecting moving objects in videos from static camera. The basic idea of the approach is that compute the difference between the current frame and background image generated by the algorithm to find out ROI (Region of Interest) for further processing. OpenCV (Open Source Computer Vision Library) provides methods such as GMG, MOG, MOG2 and KNN.

The most commonly used background subtraction method in OpenCV is MOG2, provided with OpenCL and CUDA speedup support. KNN has better accuracy than MOG2 for complex dynamic scenes but speedup isn't provided by OpenCV currently. If KNN is used in surveillance system which emphasizes on real time moving objects detection, current performance can't reach 30fps which is normal standard for the camera. Our research targets to implement parallel techniques or hardware acceleration for KNN which is not provided in OpenCV.

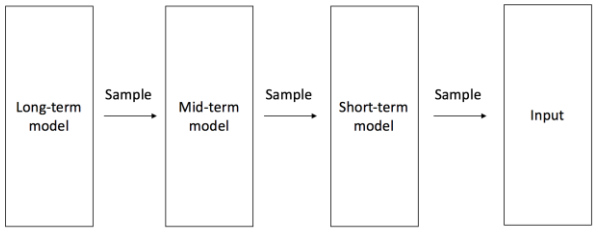
2. PROPOSED SOLUTION

The basic concept for MOG2 and KNN is use probability models to compute out the long-term distribution for value of each pixel formed by multiple frames of video. The following diagram are the common steps in both MOG2 and KNN and we will focus on parallelizing the second step because most time spent in it. Since there is no dependency for each pixel to compute its model so with this proper we consider it is possible to do operations in parallel.



In the KNN approach, it uses the k-NN algorithm to estimate whether a pixel belongs to background. The model will sample from not only the background but also the foreground. Each sample has an indicator, the indicator has a value 0 if the sample is assigned to the foreground, a value 1 if the sample is assigned to the background. It will count the close background samples. If this value is greater than the threshold the pixel is classified as background. When the model samples a pixel, it will also count all the close samples. If this value greater than the threshold the indicator will be set to 1 too.

The most important thing about parallelizing is how the model sample pixels. The model has three parts: short-term model, medium-term model, and long-term model. Each of them has different time period to sample pixels from input, and it uses a random sample in this time period.



The difficulty to parallelize is the algorithm use rand() function to determine which pixel to sample in the next time period, since rand() is not thread-safe. Using mutexes will not a good idea in this algorithm. We finally decide to make each pixel model use the same numbers from rand(), since the pixel model is independent to each other. So we just need to call rand() once before the parallel part.

3. EXPERIMENTAL METHODOLOGY

We take a static street cam video to test our performance. The video comes from one of our teammate's previous study, it is a 1920x1080, 1-min long video.

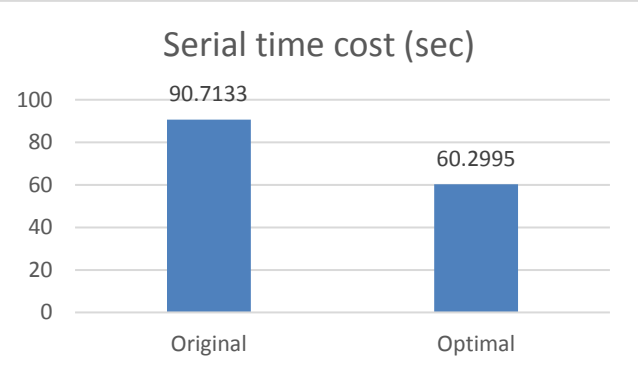
The hardware and software we use to benchmark lists below.

Table 1. Test environment

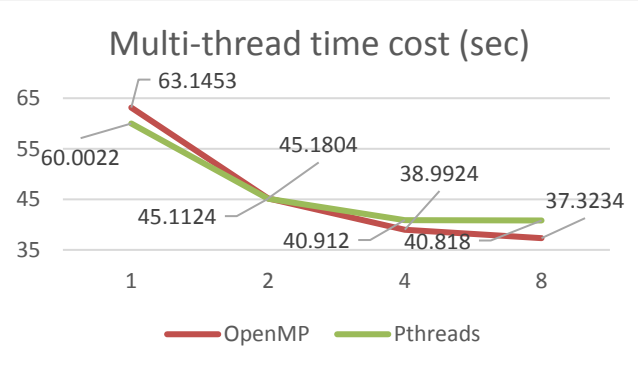
OS	CPU	GPU	OpenCV Version
Ubuntu 16.04.2	Intel Core i7-4790K	NVIDIA GTX 1060 6GB	3.3.1

4. EXPERIMENTAL RESULTS

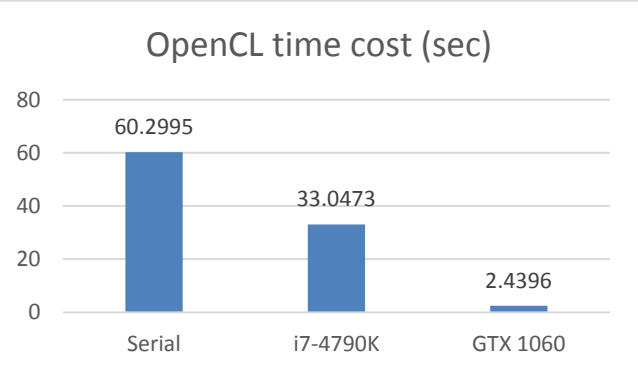
Since we change the behavior when it uses rand(), the serial program have 50% performance improvement.



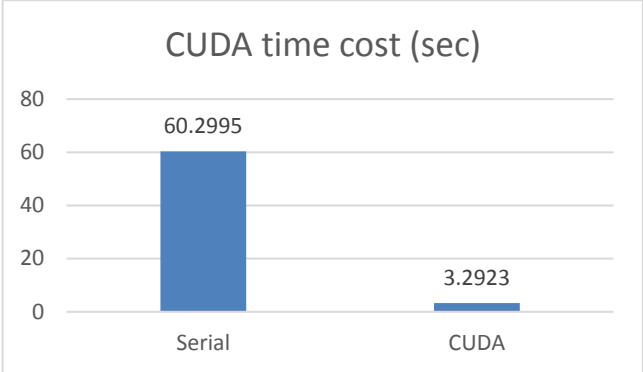
In multi-thread optimization, we compare OpenMP and Pthreads in different thread number. OpenMP version is set with a dymanic schedule.



In OpenCL optimization, we compare the difference of OpenCL on CPU and GPU platform. OpenCL on CPU still have better performance than multi-thread version.

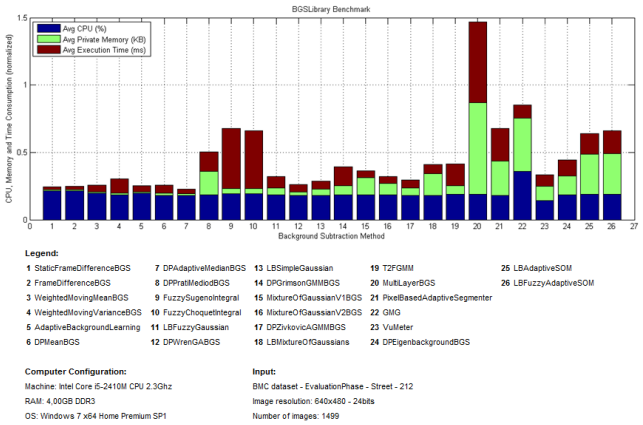


In CUDA optimization, the performance improvement is still huge, but slower than OpenCL version.



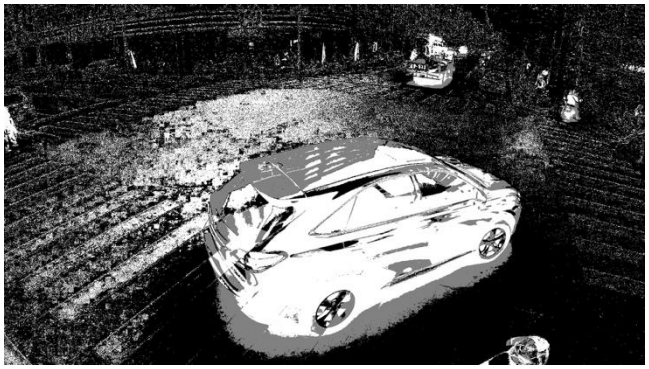
5. RELATED WORK

In [3] compares 29 methods implemented in the [4] with BMC (Background Models Challenge) dataset, which has the originality to contain both synthetic and real sequences (more precisely, it is composed of 20 synthetic videos and 9 real videos). In this benchmark, a software (BMC Wizard) is employed to compute four quality measures.

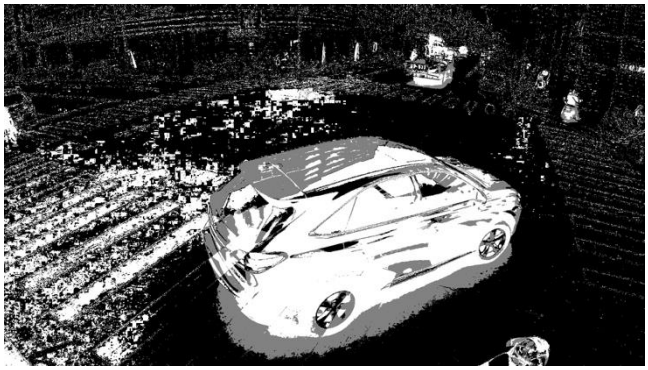


6. CONCLUSIONS

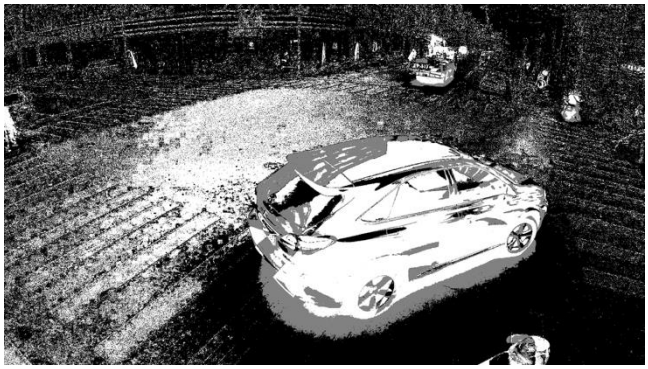
We use a variety of parallel frameworks to parallelize our target and get a great result, but there are still some problems.



This is a single frame of the output from the original program, if we take a look of the output from the program which we optimize, the result is interest.



The result shows that changing all pixel models to use same number from rand() will cause the output's nature different from the original program, so it maybe not a good idea to do that. In the version we submit to the OpenCV repository, we use OpenCV's randu() function to produce a random array at once before the parallel part, and the performance is still as good as we do in the project, so the production of the random numbers is not a big deal to the whole algorithm. And now the result is correct like the output below.



Another change to the version we submit to the OpenCV repository is that we use OpenCV's parallel_for_ framework, this framework will try to use the multi-thread framework not only OpenMP and Pthreads but also other frameworks like Intel TBB. In the end, the performance of the parallel_for_ framework is also as good as the OpenCL version, it shows the multi-thread framework with optimization is still powerful.

7. REFERENCES

[1] Zivkovic, Z., van der Heijden, F. 2006. *Efficient adaptive density estimation per image pixel for the task of background subtractio*n. In *Pattern Recognition Letters*, 27(7):773–780, 2006.

[2] Zivkovic, Z. 2004. *Improved Adaptive Gaussian Mixture Model for Background Subtraction*. *Proc.* In *Int'l Conf. Pattern Recognition*, vol. 2, pp. 28-31, 2004.

[3] Andrwes, S., Antoine V. 2014. *A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos*. In *Computer Vision and Image Understanding*, May, 2014.

[4] Andrwes, S. *bgslibrary*. Github repository https://github.com/andrewssobral/bgslibrary