CSC2231 Course Project - CUDA Accelerated ORB-SLAM2

Nu Zhang - 1008647270
nu.zhang@mail.utoronto.ca

1 Introduction

1.1 Background

Simultaneous localization and mapping (SLAM) is a method to construct a map of an unknown environment while estimating the device's location in the environment at the same time[1]. SLAM has been applied in many fields, such as indoor path planning[2], UAV navigation[3], autonomous driving[4], etc. Various sensors can be used as input for SLAM algorithms, including laser sensors, sonar, GPS, etc[5]. In particular, the SLAM algorithm that only uses the camera is referred to as visual SLAM[6]. Since mobile devices are usually equipped with cameras, visual SLAM has been widely used in the field of mobile augmented reality[7].

Many visual SLAM algorithms have been proposed, including MonoSLAM[8], PTAM[9], DTAM[10], and so on. ORB-SLAM by Mur-Atal, Montiel and Tardós[11] is a well-known and effective visual SLAM algorithm developed from PTAM. ORB-SLAM2[12] added support for stereo and RGB-D cameras on the base of ORB-SLAM. It is worth mentioning that the ORB-SLAM and ORB-SLAM2 are available as open source under GPL-v3 license[13], making it easy to access and explore.

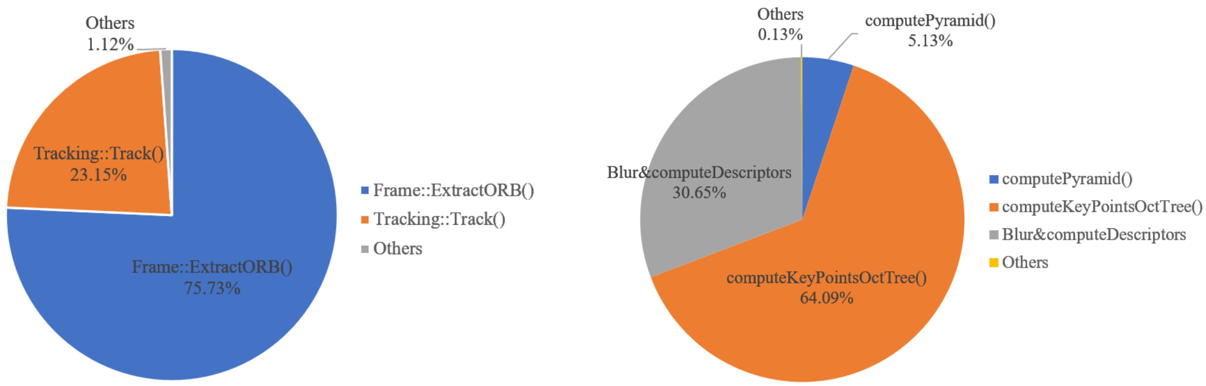
1.2 Motivation

The original ORB-SLAM algorithm[14] is designed for CPUs. With a moderate CPU, e.g. core i5, the algorithm is capable of processing images in real-time. However, for embedded platforms such as drones, and robots, which aren't equipped with fast enough CPUs, reaching real-time performance maybe difficult. Meanwhile, these embedded platforms are often equipped with onboard GPUs to perform machine learning tasks on the edge. To make better utilization of the hardware, this project aims at using GPU to accelerate ORB-SLAM2 algorithm without sacrificing accuracy. Also, I hope to develop a profound understanding of SLAM algorithms and improve general-purpose GPU programming skills in this process.

2 Performance Analysis

2.1 System Overview

The ORB-SLAM2 system contains three parallel threads, tracking thread, local mapping thread and loop closing thread[12]. The tracking thread tracks each frame, estimates camera pose, tracks the local map and generates keyframes. The local mapping thread processes keyframes and builds the map with local bundle adjustment (local BA). Local BA performs a series of operations including generating new map points, adjusting existing map points, and removing redundant keyframes. The loop closing thread detects closed loops and performs loop correction. Local mapping thread and loop closing thread are activated only when new keyframes are generated by tracking thread. And the system can only accept a new frame after tracking thread has finished processing the current frame. In general, tracking thread has a much greater impact



(a) tracking thread

(b) Frame::ExtractORB()

Figure 1: Proportion of execution time of functions in (a) tracking thread and (b) Frame::ExtractORB()

on the performance of the system. Based on this observation, this project focuses on accelerating tracking thread.

2.2 Bottlenecks Identification

To identify performance bottlenecks, the original ORB-SLAM2 algorithm is tested on a laptop with a Core i7-6700HQ and 16GB RAM. The mode is set to monocular and KITTI04[15] sequence is used as input. The execution time of different functions is recorded. As shown in Fig. 1(a), tracking thread spends more than 75% of its time on function `Frame::ExtractORB()`. And Fig. 1(b) shows that `Frame::ExtractORB()` includes three time-consuming functions, `computePyramid()`, `computeKeyPointsOctTree()` and `Blur&computeDescriptors`. To speedup the algorithm, this project utilizes CUDA to accelerate these functions.

3 Acceleration

3.1 Accelerate function `computePyramid()`

The first part of `Frame::ExtractORB()` is to generate the image pyramid by function `computePyramid()`. The image pyramid consists of `n` levels. `n` is a parameter loaded from the configuration file when the system starts. And each level is a scaled-down version of the input frame. The purpose of the image pyramid is to compensate for the problem that ORB descriptors do not have scale invariance[11].

Accelerating this function can be done by utilizing `resize()` function provided by OpenCV CUDA library. The execution time of the accelerated function is tested on a laptop with a Core i7-6700HQ, 16GB RAM, and a GTX1060 mobile GPU. The KITTI04[15] is used as input. The baseline version of the function can be found on the Github page of ORB-SLAM2[14]. And the CUDA accelerated version can be found on my Github page[16]. And the time needed to transfer data between host and device is taken into account. This experiment setup applies to all performance comparisons presented in this section. Fig. 2 shows that the speedup is not much. On the one hand, the `resize()` function of OpenCV used by the baseline is highly optimized and can achieve good performance on CPUs. On the other hand, the size of the input image is not large, and resizing is not a computation intensive task. A large portion of time is spent on data transfer between host and device, resulting in an insignificant performance improvement.

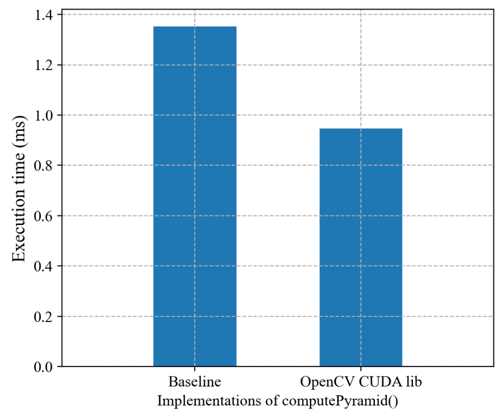


Figure 2: The execution time of `computePyramid()`

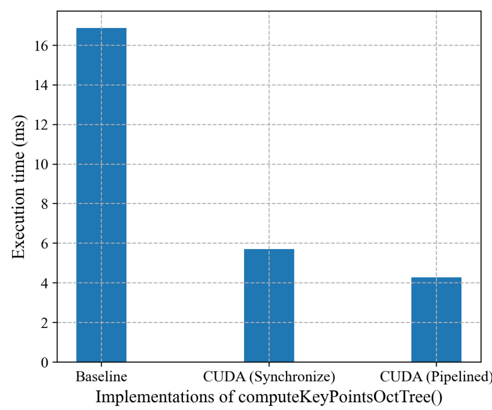


Figure 3: The execution time of `computeKeyPointsOctTree()`

3.2 Accelerate function `computeKeyPointsOctTree()`

Function `computeKeyPointsOctTree()` extracts the key points of every level in the image in the image pyramid. For each level, the image is first divided into 900 blocks. Then for each block, OpenCV's FAST detector is used with a high threshold to detect the corners. If no corners are detected, then it is detected again with a lower threshold. The purpose of this method is to ensure that the key points are evenly distributed in all parts of the image, which makes it less likely to lose tracking when the camera moves. Using OpenCV's CUDA version of the FAST detector for acceleration is feasible, but not optimal. Because deciding whether a second detection is necessary requires transferring the results of the first detection from device memory to host memory. A large number of redundant data transfers degrade the performance significantly. So the best solution is to construct a customized CUDA kernel that can generate the results with just one kernel call and one data transfer from device to host. In addition, since the computation of different levels is independent, pipelining can be used to parallelize kernel execution and data transfer. And after switching to asynchronous execution, the CPU can perform other work while the GPU is computing, improving hardware utilization.

The custom kernel `detectFAST_kernel()` is defined in src/kernel.cu[16], modified from the source code of the OpenCV CUDA library[17]. Fig. 3 shows the acceleration results, where *CUDA (Synchronize)* represents the performance of synchronized call of the custom kernel, and *CUDA (Pipelined)* represents the performance of pipelined execution of the kernel. Overall, around 3.9x speedup is achieved by this method.

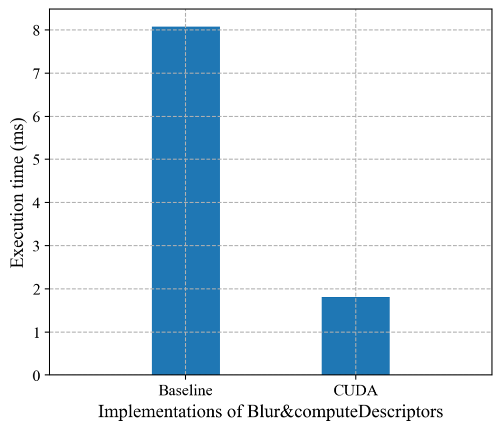


Figure 4: The execution time of `Blur&computeDescriptors`

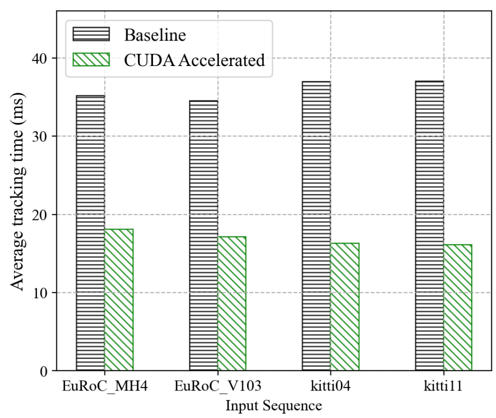


Figure 5: End to end evaluation results

3.3 Accelerate function `Blur&computeDescriptors`

`Blur&computeDescriptors` is not a separately defined function. It's a block of code that performs Gaussian blur to each level of the pyramid and calculates the descriptors of the key points detected by `computeKeyPointsOctTree()`. For the Gaussian blur part, OpenCV CUDA library can be used for acceleration. And for the part that computes the descriptors, no library functions are available. The authors of the ORB-SLAM2 wrote a function `computeOrbDescriptor()` to do the calculations. Referring to this implementation, a custom kernel is created for acceleration.

The custom kernel `computeDescriptors_kernel()` is defined in src/kernel.cu[16]. As shown in Fig. 4, this method brings 4.5x speedup to this part of the system.

4 Evaluation

End-to-end evaluation is performed with the same computer mentioned in section 3.1. Sequences from EuRoC[18] and KITTI[15] are selected as input. The results are shown in Fig. 5. It can be found that the performance improvement of the KITTI dataset is larger than that of the EuRoC dataset. This may be due to the larger image size of the KITTI dataset, which requires more time for the CPU to process. In conclusion, with all the above acceleration methods, an end-to-end speedup of 2x is achieved.

5 Conclusion

This project firstly timed each part of the ORB-SLAM2 to identify the performance bottlenecks of the algorithm. Then OpenCV CUDA library, custom kernels, and pipeline execution are used to accelerate the bottleneck functions. The accelerated algorithms are finally tested with recognized datasets. The test results show that a 2x end-to-end speedup is achieved.

Acknowledgment

This is a course project for CSC2231@UofT supervised by Professor Nandita Vijaykumar.

References

[1] Wikipedia. Simultaneous localization and mapping. https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping, 2022. [Online; accessed 18-Feb-2022].

[2] Xuexi Zhang, Jiajun Lai, Dongliang Xu, Huaijun Li, and Minyue Fu. 2d lidar-based slam and path planning for indoor rescue using mobile robots. *Journal of Advanced Transportation*, 2020, 2020.

[3] Wilbert G Aguilar, Guillermo A Rodríguez, Leandro Álvarez, Sebastián Sandoval, Fernando Quisaguano, and Alex Limaico. Visual slam with a rgb-d camera on a quadrotor uav using on-board processing. In *International Work-Conference on Artificial Neural Networks*, pages 596–606. Springer, 2017.

[4] Henning Lategahn, Andreas Geiger, and Bernd Kitt. Visual slam for autonomous ground vehicles. In *2011 IEEE International Conference on Robotics and Automation*, pages 1732–1737, 2011.

[5] Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. The slam problem: a survey. *Artificial Intelligence Research and Development*, pages 363–371, 2008.

[6] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):1–11, 2017.

[7] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. 2015.

[8] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[9] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[10] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011.

[11] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[12] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[13] J. M. M. Montiel Raúl Mur-Artal, Juan D. Tardós and Dorian Gálvez-López. ORB-SLAM. http://webdiis.unizar.es/~raulmur/orbslam/, 2016. [Online; accessed 18-Feb-2022].

[14] J. M. M. Montiel Raul Mur-Artal, Juan D. Tardos and Dorian Galvez-Lopez. ORB-SLAM2 Github page. https://github.com/raulmur/ORB_SLAM2, 2017. [Online; accessed 11-Apr-2022].

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[16] Nu Zhang. CUDA Accelerated ORB-SLAM2 Github page. https://github.com/Dunnnnn/cuda-accelerated-ORB-SLAM2, 2022. [Online; accessed 11-Apr-2022].

[17] OpenCV. OpenCV CUDA library source code. https://github.com/opencv/opencv_contrib/blob/029c2833623ae658fb1820d5da30e5023134eaf5/modules/cudafeatures2d/src/cuda/fast.cu, 2018. [Online; accessed 11-Apr-2022].

[18] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.