Brief Announcement: Optimized GPU-accelerated Feature Extraction for ORB-SLAM Systems

Filippo Muzzini
University of Modena and Reggio Emilia
Modena, Italy
filippo.muzzini@unimore.it

Nicola Capodieci
University of Modena and Reggio Emilia
Modena, Italy
nicola.capodieci@unimore.it

Roberto Cavicchioli
University of Modena and Reggio Emilia
Reggio Emilia, Italy
roberto.cavicchioli@unimore.it

Benjamin Rouxel
University of Modena and Reggio Emilia
Modena, Italy
benjamin.rouxel@unimore.it

ABSTRACT

Reducing the execution time of ORB-SLAM algorithm is a crucial aspect of autonomous vehicles since it is computationally intensive for embedded boards. We propose a parallel GPU-based implementation, able to run on embedded boards, of the Tracking part of the ORB-SLAM2/3 algorithm. Our implementation is not simply a GPU port of the tracking phase. Instead, we propose a novel method to accelerate image Pyramid construction on GPUs. Comparison against state-of-the-art CPU and GPU implementations, considering both computational time and trajectory errors shows improvement on execution time in well-known datasets, such as KITTI and EuRoC.

CCS CONCEPTS

• **Computer systems organization** → *Single instruction, multiple data*; **Embedded software**; *Robotics.*

KEYWORDS

GPU, ORB-SLAM, CUDA, Parallel

1 INTRODUCTION

Autonomous vehicles, such as Unmanned Aerial Vehicles (UAVs) and self-driving cars [9] must use sensors and algorithms to perceive the external environment to localize themselves in the world. The Simultaneous Localization and Mapping [25] (SLAM) algorithm builds a map and localizes the vehicle using sensors such as Lidar, Inertial Measurement Unit (IMU), GPS, and cameras. It maps unknown environments and localizes the vehicle when it reaches an already visited location. The ORB-SLAM [14] system exploits camera sensors, that have been thoroughly investigated due to their low price, small size, and easy setup [24]. Cameras are used to extract points as features across subsequently captured images, to then identify elements in the surrounding environment Moreover, by using stereo cameras, hence processing two concurrent images at a time, the system is able to retrieve additional information such as depth. SLAM-based systems are characterized by stringent real-time constraints as localization information must be collected before they become obsolete. Such real-time requirements can be achieved through powerful multicore CPUs, by exploiting compute accelerators, and/or by fine-tuning the algorithm parameters and implementation details. Trivially, these aspects play an important role in ORB-SLAM based approaches. That is why many versions of ORB-SLAM that exploit accelerators have been proposed in the literature, such as [8] and [12]. Our focus is on the *Tracking* phase of ORB-SLAM, as this is where most efforts from the previous work have been focused. We propose a novel and highly optimized GPU based implementation of ORB-SLAM that exploits streams to execute concurrent tasks. Moreover, we propose a novel parallel implementation for *Pyramid* construction. Lastly, we compare our implementations with respect to other state of art methods and we release publicly the source code of our implementation.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND AND RELATED WORK

GPUs were initially designed with the goal of accelerating graphic workloads. However, they can be exploited for general purpose computing (GPGPU). A GPU is a SIMD processor (Single Instruction Multiple Data) able to process large amounts of data in a parallel fashion. Since the GPU has its private memory space, it is necessary to copy data to this memory in order to perform the computation (called *kernel*) In order to assist the programmer in exploiting the GPU capabilities, NVIDIA released a proprietary programming model called CUDA (Compute Unified Device Architecture). CUDA enables to dispatch *kernels* through a launch configuration, i.e. a grid specified by the programmer in which parallel GPU threads are logically organized [4] in blocks of threads. Moreover, it allows the programmer to express an added layer of parallelism through CUDA *Streams* and *Events*. Commands (compute kernel invocations and memory movements) among different *Streams* are able to execute concurrently[16] and can synchronize the execution using *Events*. ORB-SLAM can be categorized in the *Visual SLAM* category of SLAM systems, in particular in the *features based* approaches since it uses the image features to estimate the system pose. Other approaches are *Direct* methods that use the image's pixel without extracting features [6, 27] and *keyframe based* methods that select only a few *keyframe* to reconstruct the map [11, 22]. ORB-SLAM systems have evolved over years. The first version [14] introduces the use of ORB descriptor [20] as features; in ORB-SLAM2 [15], the authors introduce the use of RGB-D cameras; finally in ORB-SLAM3 [3]. ORB-SLAM features a considerable computational time so some works tried to reduce this by exploiting accelerators. In [10], a variety of acceleration methods for ORB-SLAM or more generic *Visual Odometry*, are compared; in [13], the authors accelerated the method using OpenCL for FPGA and NVIDIA platforms; in [1, 12], CUDA is exploited to accelerate ORB-SLAM2 on NVIDIA platforms using CUDA and OpenCV[1] to extract and match features in Stereo camera systems. Moreover, in [1], the authors use OpenVX to offload computation to GPU. In [8], the authors partially implemented the features extraction phase using CUDA and use OpenCV to perform the image scaling. Our proposal improves the parallelism and concurrency design of ORB-SLAM2 and ORB-SLAM3 algorithms. In particular, with respect to the previous works, we propose a different image scaling approach, we maximize the GPU utilization using *Streams* and custom *kernels*, and we reduce the data copies between CPU and GPU.

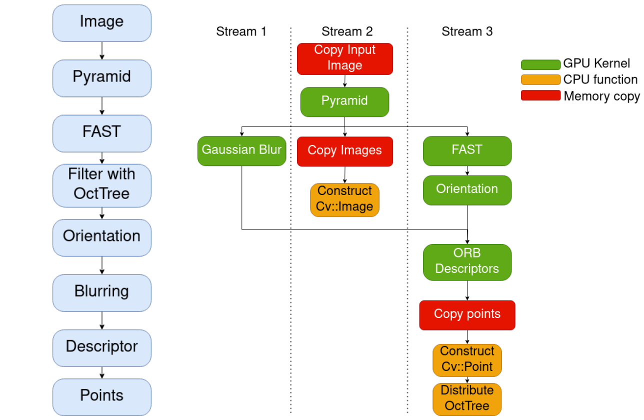
3 ORB-SLAM OVERVIEW

The interested reader can find the various ORB-SLAM phases in [14]. We focus on the optimization of *Tracking* part, in particular *ORB Extraction* phase (Figure 1a), which is composed of six steps: *Pyramid*, *FAST*, *Distribute Octree*, *Orientation*, *Gaussian Blur* and *ORB Descriptor*. The *Pyramid* construction produces, from an input image, several versions of that image at different levels of scaling [23]. Originally, the input image was sequentially scaled: each level is constructed using the precedent one. Then, the FAST algorithm [19] detects *corners* within the image pixels. A corner is a pixel that displays a significantly different luminance value compared to its neighboring pixels. Since the number of points computed by *FAST* can be huge, the *Octree Distribution* [17] algorithm is performed as a point filter. It preserves isolated points and prunes less significant ones in the dense areas of the image. In other words, the filter attempts to prune as many points as possible, while still guaranteeing the minimum points required by the system. Then, for each point, the orientation is computed using the *intensity centroid* [18]. Finally, all images are blurred and the ORB Descriptor (ORB-D) is computed for each point. ORB-D is based on BRIEF descriptors [2] that have the limitation of not considering point orientation so they will fail to recognize the same point across two or more frames. The authors in [20] introduced a correction to consider the previously computed orientation.

4 OUR NOVEL IMPLEMENTATION

We provide a novel GPU-based implementation of the *ORB Extraction* phase (see Figure 1b) of the ORB-SLAM pipeline in order to achieve improved performance with regard to its computational time. More specifically, we use three *Streams* to execute concurrently three independent flows: (1) one to compute the blurred images; (2) one to execute Pyramid and related CPU/GPU memory transfers; (3) one to execute the feature extraction algorithms and final memory transfer; we wrote *kernels* for the *Pyramid*, *Gaussian Blur*, *FAST*, *Orientation*, and *ORB Descriptors*; and we reduce the

[1]https://opencv.org



(a) Sequential Flow **(b) Concurrent Flow**

Figure 1: ORB Extraction flows

memories copies between CPU and GPU. The *Distribute Octree* phase is not suitable for GPU implementation. *Gaussian Blur* and *FAST* tasks can be parallelized mapping one thread for each image's pixel since the computation of a pixel is independent of others. *Orientation* and *ORB Descriptor* task are computed over the points extracted by *FAST* and each point is independent of others so they can be computed in a parallel manner. Differently from the baseline image pyramid construction, our kernel simultaneously computes each level starting from the original image. Each pixel of each level is assigned to a CUDA GPU thread. Each thread computes its pixel value starting from the original image, so to remove dependencies among the pyramid levels and to have a result qualitatively equal to the baseline. Considering the scaling factor f of the current level, the reference pixel coordinates from the original image are computed as in eq. (1).

$$x_{up} = \lceil x * f \rceil,\ x_{low} = \lfloor x * f \rfloor\ y_{up} = \lceil y * f \rceil,\ y_{low} = \lfloor y * f \rfloor \quad (1)$$

Using the reference pixels, the new pixel value is computed by applying the *Bilinear Interpolation* [21]. Ultimately, the difference between the traditional method of *Pyramid* construction and our approach lies in the selection of reference pixels.

To reduce the amount of memory copies between CPU and GPU the *Distribute Octree* phase is moved after the ORB-D phase, this allows having only one memory copy after *ORB-D* instead of two (one after *FAST* and one before *Orientation* since *Distribute Octree* runs on CPU and other phases on GPU. This means that *Orientation* and *ORB-D* phases must process more points but the effort is less than having more memory copies [5].

5 EXPERIMENTS AND RESULTS

We performed experiments on the Nvidia Xavier AGX board (512 NVIDIA cores, 8 cores ARM CPU, 32GB RAM). We compared our implementation with the open source implementations of the original *ORB-SLAM2* [15][2], *ORB-GPU* [8][3], and *ORB-dataflow* [1][4]. While

[2]https://github.com/raulmur/ORB_SLAM2
[3]https://github.com/yunchih/ORB-SLAM2-GPU2016-final
[4]https://github.com/xaldyz/dataflow-orbslam

Table 1: ORB-SLAM2 results for KITTI dataset

		Stereo			Monocular			
		ORB-SLAM2	ORB-GPU	*Our*	ORB-SLAM2	ORB-GPU	ORB-dataflow	*Our*
KITTI04	ATE(m)	0.805	**0.435**	0.571	1.645	0.552	0.633	**0.495**
	mean time(s)	113.471	76.560	**67.189**	49.039	20.568	31.556	**17.870**
	min time(s)	101.737	68.430	**58.278**	42.156	14.983	33.752	**11.935**
	max time(s)	178.186	**110.765**	137.335	145.387	141.620	**44.609**	131.922
KITTI06	ATE(m)	2.688	1.440	**1.391**	**13.443**	16.172	16.229	22.953
	mean time(s)	92.102	81.771	**72.370**	49.719	23.369	70.273	**17.979**
	min time(s)	73.580	65.330	**53.982**	40.599	15.053	12.405	**10.601**
	max time(s)	391.772	**341.873**	453.502	207.565	214.655	**194.517**	235.658
KITTI07	ATE(m)	0.871	**0.805**	1.380	4.299	**4.260**	4.438	4.768
	mean time(s)	87.546	79.150	**70.828**	50.077	21.559	67.521	**17.887**
	min time(s)	72.527	67.656	**57.499**	39.392	14.271	**12.372**	12.447
	max time(s)	**238.742**	360.060	424.848	260.271	240.451	276.189	**229.278**

Table 2: ORB-SLAM3 results for EuRoC dataset

	Stereo				Monocular			
	ATE(m)	mean time(s)	min time(s)	max time(s)	ATE(m)	mean time(s)	min time(s)	max time(s)
ORB-SLAM3 [3]	**0.037**	57.068	44.941	**62.634**	3.059	29.555	19.388	**118.630**
Our	0.064	**33.021**	**21.843**	65.219	**1.101**	**19.095**	**11.720**	441.696

ORB-SLAM2 is fully implemented on the CPU, *ORB-GPU* and *ORB-dataflow*, derived from it, exploit the GPU acceleration. We used sequences 04, 06, and 07 from the Kitti dataset [7] as test scenarios (image size: 1226x370px). The first sequence mostly has straight trajectories, the second features a large number of close turns, and the last represents an urban scenario. For all these scenarios, we measured the computational time needed by the *Tracking phase* to process a single frame in the *Monocular* and *Stereo* versions. We also measured the Absolute Trajectory Errors (ATE) using the tool released in [26]. We also tested our implementations using the *ORB-SLAM3* [3][5] as a code base, and compared it against the original version of *ORB-SLAM3* that runs on CPU. In this case, we used the first sequence of the EuRoC Dataset (image size: 752x480px), which is a scenario designed for autonomous drones. In this case, we used the tool released with the code of *ORB-SLAM3* to measure the ATE. Table 1 reports the results for ORB-SLAM2 on the KITTI dataset in the *Monocular* and *Stereo* versions. *ORB-dataflow* does not support *Stereo* mode so we have not reported it. We can see that our implementation presents the best mean time in all situations. Considering the mean time in the *Stereo* mode, with respect to the *ORB-SLAM2* original implementation, our implementation has a speedup of 1.7x, 1.28x, and 1.24x in KITTI04, KITTI06, and KITTI07, respectively. In the *Monocular* case, for the same sequences, the speedup is 2.74x, 2.77x, and 2.80x. Regarding tracking errors, for the *Monocular* version, our implementation shows the best results for sequence 04 and in sequence 07 the error is comparable with respect to the other implementations. In sequence 06, larger error highlights the difficulty in tracking on turns. This error is mitigated in the *Stero* version in which our implementation is competitive in all sequences. Table 2 reports the results errors for the *Monocular* and *Stereo* versions for ORB-SLAM3 on the EuRoC dataset. The speedup of our implementations with respect to the original *ORB-SLAM3* is 1.55x (*Monocular*) and 1.73x (*Stereo*). In terms of trajectory errors, our implementation shows the best results in *Monocular* case and comparable in the *Stereo* case. To conclude, our implementation shows a better execution time in all situations reducing the percentage of *ORB extraction* phase over the entire *Tracking* part from 45% to 29%. The difference in trajectory error can lead to the different *Pyramid* construction since it produces less blurred images but this difference is mitigated in the *Stereo* version.

[5] https://github.com/UZ-SLAMLab/ORB_SLAM3

6 CONCLUSION

We proposed a new GPU-based implementation for the *ORB extraction* part of the ORB-SLAM *Tracking* phase we release it[6]. The presented evolution includes a new *Pyramid* construction algorithm; an optimal use of CUDA *Streams* that minimizes memory copies. We compared our implementations with SotA versions ORB-SLAM2/3, and a GPU-enabled ORB-SLAM2 version demonstrating that it outperforms the others in terms of computational time without significant precision deterioration. In future research, we plan to replace the *Distribute Octree* phase with an algorithm suitable for GPU implementation.

ACKNOWLEDGMENTS

This work was partly supported by ECSEL Joint Undertaking in H2020 project IMOCO4.E, grant agreement No.101007311.

REFERENCES

[1] Stefano Aldegheri, Nicola Bombieri, Domenico D. Bloisi, and Alessandro Farinelli. 2019. Data Flow ORB-SLAM for Real-time Performance on Embedded GPU Boards. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 5370–5375. https://doi.org/10.1109/IROS40897.2019.8967814

[2] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. 2010. BRIEF: Binary Robust Independent Elementary Features. In *Computer Vision – ECCV 2010*, Kostas Daniilidis, Petros Maragos, and Nikos Paragios (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 778–792.

[3] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. Montiel, and Juan D. Tardós. 2021. ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM. *IEEE Transactions on Robotics* 37, 6 (2021), 1874–1890. https://doi.org/10.1109/TRO.2021.3075644

[6] https://git.hipert.unimore.it/fmuzzini/cuda-accelerated-orb-slam

[4] Nicola Capodieci, Roberto Cavicchioli, and Andrea Marongiu. 2021. A taxonomy of modern GPGPU programming methods: on the benefits of a unified specification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 6 (2021), 1649–1662.

[5] Nicola Capodieci, Roberto Cavicchioli, Ignacio Sañudo Olmedo, Marco Solieri, and Marko Bertogna. 2020. Contending memory in heterogeneous SoCs: Evolution in NVIDIA Tegra embedded platforms. In *2020 IEEE 26th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 1–10.

[6] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*. Springer, 834–849.

[7] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

[8] Zhaowei Huang, Yunzhi Chen, and Yiyou Jiang. 2016. *ORB-SLAM2 GPU Optimization.* http://http://yunchih.github.io/ORB-SLAM2-GPU2016-final/

[9] Rasheed Hussain and Sherali Zeadally. 2018. Autonomous cars: Research results, issues, and future challenges. *IEEE Communications Surveys & Tutorials* 21, 2 (2018), 1275–1313.

[10] Jinwoo Jeon, Sungwook Jung, Eungchang Lee, Duckyu Choi, and Hyun Myung. 2021. Run Your Visual-Inertial Odometry on NVIDIA Jetson: Benchmark Tests on a Micro Aerial Vehicle. *IEEE Robotics and Automation Letters* 6, 3 (2021), 5332–5339. https://doi.org/10.1109/LRA.2021.3075141

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

[12] Zhenzhen Li, Lei Zhang, Chenglong Xu, Fengshan Zou, Jilai Song, and Daji Tian. 2019. An Acceleration Method Using CUDA based on ORB-SLAM2. In *2019 IEEE 9th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. 1441–1446. https://doi.org/10.1109/CYBER46603.2019.9066525

[13] Mamri, Ayoub, Abouzahir, Mohamed, Ramzi, Mustapha, and Latif, Rachid. 2021. ORB-SLAM accelerated on heterogeneous parallel architectures. *E3S Web Conf.* 229 (2021), 01055. https://doi.org/10.1051/e3sconf/202122901055

[14] Ra'ul Mur-Artal, J. M. M. Montiel, and Juan D. Tard'os. 2015. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics* 31, 5 (2015), 1147–1163. https://doi.org/10.1109/TRO.2015.2463671

[15] Ra'ul Mur-Artal and Juan D. Tard'os. 2017. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262. https://doi.org/10.1109/TRO.2017.2705103

[16] Ignacio Sañudo Olmedo, Nicola Capodieci, Jorge Luis Martinez, Andrea Marongiu, and Marko Bertogna. 2020. Dissecting the CUDA scheduling hierarchy: a performance and predictability perspective. In *2020 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 213–225.

[17] Stefan Peters. 2013. Quadtree- and octree-based approach for point data selection in 2D or 3D. *Annals of GIS* 19, 1 (2013), 37–44. https://doi.org/10.1080/19475683.2012.758171 arXiv:https://doi.org/10.1080/19475683.2012.758171

[18] Paul L Rosin. 1999. Measuring corner properties. *Computer Vision and Image Understanding* 73, 2 (1999), 291–307.

[19] Edward Rosten and Tom Drummond. 2006. Machine Learning for High-Speed Corner Detection. In *Computer Vision – ECCV 2006*, Aleš Leonardis, Horst Bischof, and Axel Pinz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 430–443.

[20] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544

[21] P.R. Smith. 1981. Bilinear interpolation of digital images. *Ultramicroscopy* 6, 2 (1981), 201–204. https://doi.org/10.1016/0304-3991(81)90061-9

[22] Hauke Strasdat, José MM Montiel, and Andrew J Davison. 2012. Visual SLAM: why filter? *Image and Vision Computing* 30, 2 (2012), 65–77.

[23] Magnus Strengert, Martin Kraus, and Thomas Ertl. 2006. Pyramid methods in GPU-based image processing. *Proceedings vision, modeling, and visualization 2006* (2006), 169–176.

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

[25] Sebastian Thrun. 2007. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*. Springer, 13–41.

[26] Huangying Zhan, Chamara Saroj Weerasekera, Jiawang Bian, and Ian Reid. 2019. Visual Odometry Revisited: What Should Be Learnt? *arXiv preprint arXiv:1909.09803* (2019).

[27] Jon Zubizarreta, Iker Aguinaga, and Jose Maria Martinez Montiel. 2020. Direct sparse mapping. *IEEE Transactions on Robotics* 36, 4 (2020), 1363–1370.