Parallel Harris Corner Detection on Heterogeneous Architecture

Yiwei He[1], Yue Ma[2], Dalian Liu[3⋆] and Xiaohua Chen[4]

[1] School of Computer and Control Engineering,
University of Chinese Academy of Sciences, Beijing, China
[2] School of Mathematical Sciences,
University of Chinese Academy of Sciences, Beijing, China
[3] Department of Basic Course Teaching,
Beijing Union University, Beijing, China
[4] Dean's office, Beijing Union University, Beijing, China

Abstract. Corner detection is a fundamental step for many image processing applications including image enhancement, object detection and pattern recognition. Recent years, the quality and the number of images are higher than before, and applications mainly perform processing on videos or image flow. With the popularity of embedded devices, the real-time processing on the limited computing resources is an essential problem in high-performance computing. In this paper, we study the parallel method of Harris corner detection and implement it on a heterogeneous architecture using OpenCL. We also adopt some optimization strategy on the many-core processor. Experimental results show that our parallel and optimization methods highly improve the performance of Harris algorithm on the limited computing resources.

Keywords: Harris Corner Detection, Heterogeneous Architecture, Parallel computing, OpenCL.

1 Introduction

Corner detection is an important problem in many image processing applications including edge detection, object detection and pattern recognition [1]. It is a fundamental step in image processing. Recent years, with the development of embedded devices or high-performance computing, the real-time computing plays a crucial role in many applications, such as video game, communication app and media player. Especially in the area of computer vision, applications always require that the system can be request clients in a few seconds. As an indispensable corner detection algorithm, Harris corner detector has been successfully used in the image processing [25], such as feature selection or edge detection. It is also accelerated based on different strategy or various compute

⋆ Corresponding author. E-mail addresses: heyiwei16@mails.ucas.ac.cn (Y. He), mayue115@mails.ucas.ac.cn (Y. Ma), ldlluck@sina.com (D. Liu), lytxiaohua@buu.edu.cn (X. Chen)

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

devices. However, much of them ignore the limitations of computing resources like embedded device, and they do not fully take advantage of the heterogeneous architecture.

Over past decades, the performance of computing device has achieved a significant development. Many large-scale computing tasks are benefited from modern processors like GPU, CPU or FPGA. Especially growing in many-core processors, massive algorithms have been parallelled and implemented on the many-core processor which could improve the efficiency of computing [11]. The general purpose computing on GPU pushed the revolution of many applications like machine learning, and more and more algorithms are transplanted to the many-core compute platforms. GPU also push the improvement of machine learning research. Many methods would be benefited from the high-performance of GPU [22, 21, 20, 15, 10, 7, 19, 24].

However, large-scale computing task is suitable for the host or server devices. For the embedded devices, the limited computing resources cannot satisfy the complexity of massive data processing or the real-time reaction. For example, some image applications on the Android or IOS which should be reacted in a few seconds. Thus, how to fully utilize the limited computation resource is a key problem which is needed to solve urgently. Two types of strategy are used to speed up. One is reducing the complexity of an algorithm, and the other is optimizing based on the architecture of computing device. In real applications, the implementation is always combined this two idea to optimize the software.

In this paper, we parallel the Harris corner detection algorithm and implement it in an environment of heterogeneous architecture which is composed of many-core and multi-core processors. We also adopt some optimization for methods basing on this unique design. We implement the algorithm by OpenCL, which is an open source parallel library working for heterogeneous architecture and it is commonly used in cross computation platforms. Experimental results prove that our implementation is accuracy and efficiency. The rest paper is organized as follow: Section 2 introduces the background and Harris corner detection, Section 3 makes an instruction of heterogeneous architecture under the cross-platform software library OpenCL and the related work of parallel Harris corner algorithm implementation. Section 4 introduces details of our implementation and optimization. Section 5 lists the accuracy of detection and computing efficiency. At last, we give the conclusion and explanation.

2 Background of Harris Corner Detection

Harris corner detector is developed basing on Moravec corner detection to mark the location of corner points precisely [5]. It is a corner detection operator which is widely used in computer vision algorithms to extract corners and infer features of an image [23]. It also contributes to the area of computer vision [8]. At the rest of this section, we give an overview of the formulation of the Harris corner detection and its algorithm.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

A corner is defined as the intersection of two edges. The main idea of Harris algorithm is that the corner would emerge when the value of an ROI (region of interest) variant dynamically with the shift to nearby regions [2]. The algorithm set a window scan the ROI in all directions; if it has a high gradient, we can infer that there may be corners in this region. We define $I(x,y)$ as a pixel in the input image, (u,v) is the offset of shifted region from the ROI. $w(x,y)$ is represented a convolution function which is Gaussian filter here. The function of the variable is defined as follow:

$$E(u,v) = \sum_{x,y} w \otimes (x,y) \left[I(x+u, y+v) - I(x,y) \right] \tag{1}$$

where $\otimes$ is represented as a convolution operator. And then we make an approximation with shifted ROI value based on Taylor series expansion equation.

$$I(x+u, y+v) \approx I(x,y) + I_x(x,y)\, u + I_y(x,y)\, v \tag{2}$$

By substituting (2) into (1) and approximate the result can be converted to matrix form:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \sum_{x,y} w(x,y) \otimes \begin{bmatrix} I_x^2(x,y) & I_x(x,y) I_y(x,y) \\ I_x(x,y) I_y(x,y) & I_y^2(x,y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{3}$$

$$= \begin{pmatrix} u & v \end{pmatrix} w(x,y) \otimes \boldsymbol{M} \begin{pmatrix} u \\ v \end{pmatrix} \tag{4}$$

The matrix H which named Harris matrix is defined as:

$$H = \sum_{x,y} w(x,y) \otimes \begin{bmatrix} I_x^2(x,y) & I_x(x,y) I_y(x,y) \\ I_x(x,y) I_y(x,y) & I_y^2(x,y) \end{bmatrix} \tag{5}$$

To determine whether the pixel is a corner point or not, we need to compute pixel criterion score $c(x,y)$ for each pixel. The function is given by

$$c(x,y) = det(H) - k\,(trace(H))^2 \tag{6}$$

$$= \lambda_1 \lambda_2 - k\,(\lambda_1 + \lambda_2)^2 \tag{7}$$

where λ_1, λ_2 are the eigenvalues of the Harris matrix H. At the last step, we calculate the criterion score $c(x,y)$ for each pixel, if the score higher than the threshold and it is the maximum value in the scan area, we mark this pixel as a corner point. The description of Harris corner detection algorithm is list in Algorithm 1.

3 Heterogeneous Architecture and Related Work

3.1 Heterogeneous Architecture

Since the improving requirement of complexity for large-scale computing, the performance of processors become more efficiently. Many-core and multi-core

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

Algorithm 1 Harris Corner Detection

Require: Input image I parameter k,
Ensure: optimal $\boldsymbol{\alpha}$ and $\boldsymbol{M}$
1: Compute image gradient I_x and I_y for every pixel;
2: Compute the element in the Harris Matrix H
3: **repeat** Each pixel
4: Define ROI of pixel by Gaussian filter
5: Update Harris matrix $\boldsymbol{H}$
6: Compute eigenvalutes of Harris matrix $\boldsymbol{H}$
7: Compute corner score of the pixel
8: **until**
9: Threshold corner score
10: Mark pixel as corner point for maximum corner score

processors make a significant contribution to many fields [9]. CPU specialize in logic operation, and contrast, GPU does well in float or integer computing. These two kinds processors cooperate each other to enhance the computing speed. This structure of CPU-GPU is a typical kind of heterogeneous architecture. Fig. 1 shows an example of heterogeneous architecture.

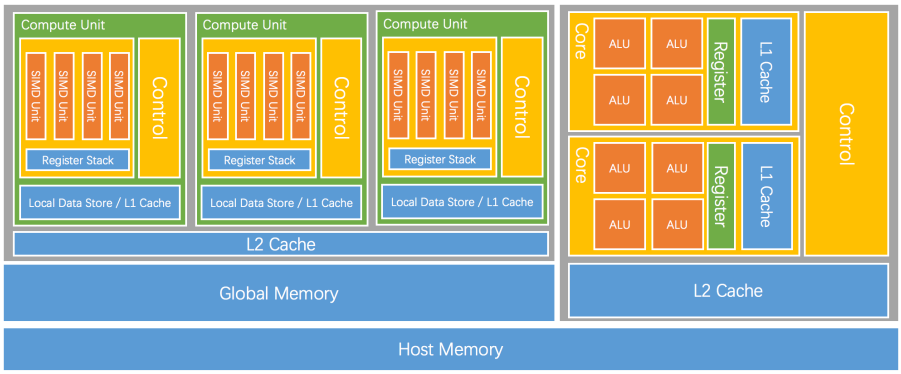

Fig. 1. Multi-core and Many-core heterogeneous architecture. There are several compute units in the GPU and each of them contains SIMD (single instruction multi data) unit, register stack and local data store. Most square of CPU is used to be memory, like Cache and register.

However, some factors limit the development of processors, including memory access and power wall, particularly the finite square of the chip for the requirement of embedded devices. With the popularity of embedded devices, the square wall of a chip is a limitation. Thus, how to fully utilize resource on-chip, like register, local memory and compute units, is a critical problem in future.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

In this paper, we consider the heterogeneous architecture, which is composed of a GPU and a CPU. For implementation, we adopt a parallel open source library named OpenCL that can be performed on various devices. It is a popular framework for programming in the heterogeneous environment. It abstracts compute devices into the same structure and constructs a communication function among compute units or devices. The most advantage of OpenCL is cross-platform. Fig. 2 shows the abstract structure in OpenCL.

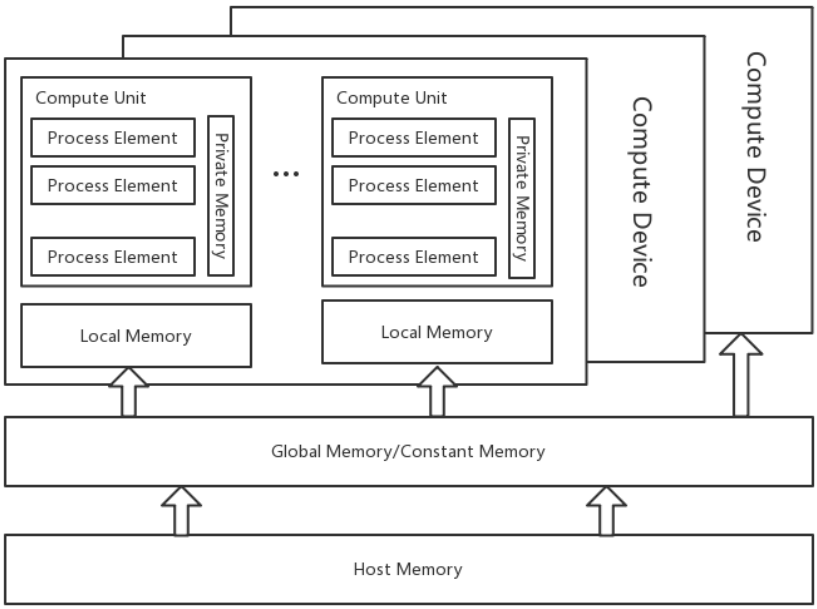

Fig. 2. OpenCL open source library abstracts computing devices in a unified framework [18]. The compute units are organized in clusters, compute devices are highest level contain several compute units which are composed by dozens of process element. Memory resources are organized in a multi-level style. The nearest from process elements are register, then in the order of local memory, global memory and host memory.

3.2 Related Works

Corner detection techniques are being widely used in many computer vision applications for example in object recognition and motion detection to find suitable candidate points for feature registration and matching. High-speed feature detection is a requirement for many real-time multimedia and computer vision

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

applications. Harris corner detector (HCD) as one of many corner detection algorithm has become a viable solution for meeting real-time requirements of the applications. There are many works to improve the efficiency of the algorithm, and some parallel implementations has been developed on different platforms. In previous work, several implementation have been proposed which target a specific device or some particular aspects of the algorithm. Saidani et al. [16] used the Harris algorithm for the detection of interest points in an image as a benchmark to compare the performance of several parallel schemes on a Cell processor. To attain further speedup, Phull et al. [13] proposed the implementation of this low complexity corner detector algorithm on a parallel computing architecture, a GPU software library namely Compute Unified Device Architecture (CUDA). Paul and his co-author [12] present a new resource-aware Harris corner-detection algorithm for many-core processors. The novel algorithm can adapt itself to the dynamically varying load on a many-core processor to process the frame within a predefined time interval. The HDC algorithm was implemented as a hardware co-processor on the FPGA portion of the SoC, by Schulz et al. [17]. Tadonki et al. [3] study a direct and explicit implementation of common and novel optimization strategies, and provide a NUMA-aware parallelization. Moreover, Jasani et al. [6] proposed a bit-width optimization strategy for designing hardware-efficient HCD that exploits the thresholding step in the algorithm. Xiao Han et al. [4] implement the HCD using OpenCL and perform it on the desktop level GPU and gain a 77 times speedup.

4 Harris Corner Detection OpenCL Implementation

In this section, we introduce our strategy of parallelization for Harris corner detection in OpenCL implementation. As shown in Section 2, there are many operators based on the pixel level. Thus, we design our parallel implementation in pixel grain size. We parallel the step of Gaussian blur convolution, Gradient X, Y computing and Harris matrix construction which are implemented on GPU. The step of eigenvalues computes and corner response are implemented on CPU.

We divide algorithm into two kernel function. One is the construction of Harris matrix, and another is pixel score. Compared with other implementation, we decrease the number of the kernels. We integrate the function into one kernel as far as possible for the reason that it can reduce the time of communication between host and kernel device, like host memory and graphics memory. It also increases the ratio of data reuse and speeds up the program. In our design, we assume that the computing resource is limited, such as register, shared memory or computing unit, and our primary target is speeding up our program in the limited resource.

4.1 Kernel of Convolution and Matrix Construction

The compute of Gaussian blur convolution, image gradient and Harris matrix are merged into one kernel. For this kernel, we construct a computing space

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

which is the same dimension as an input image. Every thread deals a pixel task and output one Harris matrix. All outputs in threads compose a complete Harris matrix. For a thread In this kernel, we first compute the gradient X $I_x(x, y)$and gradient Y $I_y(x, y)$ of this pixel and then compute its own $I_x^2(x, y)$, $I_y^2(x, y)$ and $I_x(x, y) I_y(x, y)$. Finally, we use the operation of Gaussian blur convolution to filter the pixel with its neighbourhood. The procedure description of this kernel is shown in Fig. 3.

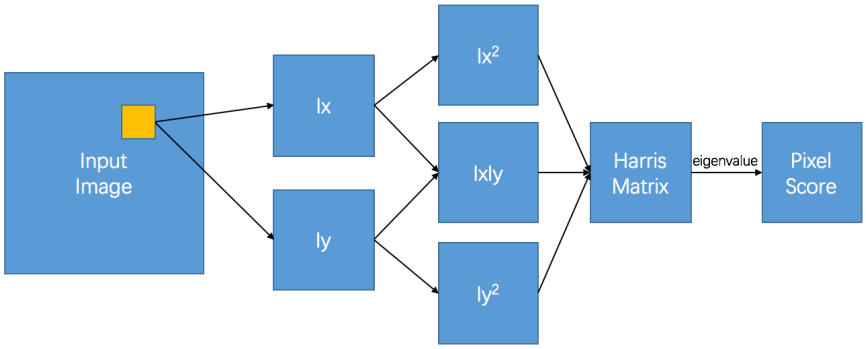

Fig. 3. The figure indicates the process for the algorithm of Harris corner detection.

Optimization strategy: The pixel level computing is beneficial for many-core architecture since its high parallelism and numerical value compute. We utilize this advantage of convolution that every thread compute a mask filter. However, in the process of convolution or gradient compute, it exists many memory access. It is low efficiency when read data from global memory to compute unit frequently. To solve this problem, we move pixels nearby target to shared memory on-chip first. This method could improve the local data repetition rate and make computing units access data which are stored in the consecutive address, namely combination access. In our implementation, we set the local pixel to the size of local computing space.

4.2 Kernel of Corner Response

After the first kernel computing, we get the corner score for every pixel in the ROI which we defined. These corner scores can report the probability of a corner point existing in the corresponding ROI. If a corner score is a negative value, it means there may be an edge in this region, and a small value indicates this area may be a flat region. Thus, we need to get the score values which are larger than the threshold, which indicate that there exists a corner in the ROI of this pixel. At last, we adopt the non-maximum suppression (NMS) stage which is aim to get the local maximum value. We set the pixel which have local maximum value as a corner point.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

In our implementation, we fix a $3 * 3$ window to search the neighborhood nearby the pixel. Every thread in the computing space is assigned a $3 * 3$ region, and if the corner score is larger than the threshold and it is the maximum value of this region, we set this pixel as a corner point. Similar to kernel convolution, we store consecutive data together from global memory to the local data memory on-chip. For limited store resource like register, we prefer the search window as little as possible.

5 Experimental Results

In this section, we will introduce experimental results for our implementation regarding accuracy and effectiveness on our heterogeneous hardware architecture.

5.1 Detection Accuracy

We use the function *HarrisCorner* in *OpenCV* as our benchmark of serial implementation. OpenCV is an open source software library, and it is utilized in image processing and computer vision. Similar with OpenCL, it can take advantage of the cross-platform and hardware acceleration based on heterogeneous compute device [14]. Fig. 4 shows the results of corner detection.

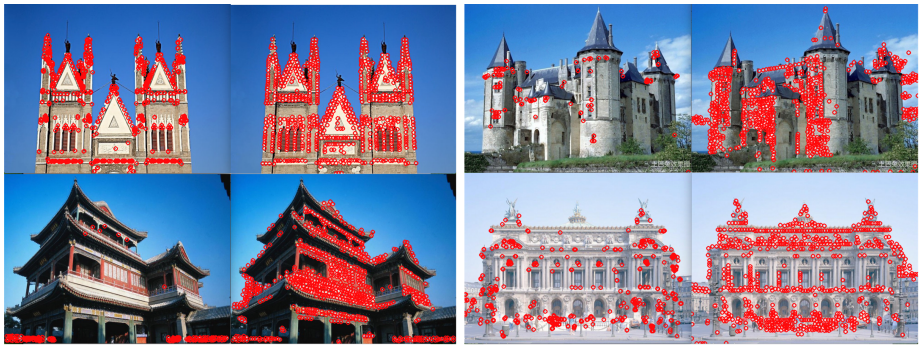

Fig. 4. The experimental results are shown in this figure. The corners detected by algorithms are in the red circles. The left image for each of sub-images is the detection result of baseline method, which is the function in OpenCV. The right image for each of sub-images is the results of our paralleled method. Contrast, our method is more stable and more precisely.

5.2 Performance Results

To evaluate our implementation, we perform our experiments on MacOS with OpenCL 1.2. The hardware configure is a CPU of 2.6GHz Intel Core i5 and

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

a many-core processor namely Intel Iris. Iris is a lightweight GPU with limited compute units and memory, which provides 40 stream processors. It is a typically many-core processor with limited computing resource.

Comparing with OpenCV function *HarrisCorner*, our implementation (image size : 640×480) on the CPU-GPU architecture could get speedup of 11.7. With the ROI increasing, the speedup is improved. It proves that our design is efficiency. The experimental results are lists in Table.1.

Table 1. We change the size of ROI to test the compute time. This table list the compute time on CPU and heterogeneous device. When the size of ROI augment, the speedup is increasing.

Size of ROI	**CPU time (ms)**	**Heterogeneous time (ms)**	**Speedup**
3×3	120.34	11.05	10.89
5×5	144.10	10.94	13.17
7×7	147.43	11.09	13.29
Average	137.29	11.03	12.45

6 Conclusion

In this paper, we have paralleled the Harris corner detection algorithm and implemented it on the heterogeneous architecture using OpenCL. Our implementation has achieved an acceleration compared with open library function in OpenCV. Our design considers the utilization of memory resource. It increases memory reuse ratio as possible. We implement Harris corner detection on a limited resource device and gain a speedup.

Acknowledgments

This work has been partially supported by grants from the National Natural Science Foundation of China (Nos. 61472390, 71731009, 71331005 and 91546201), the Beijing Natural Science Foundation (No.1162005), Premium Funding Project for Academic Human Resources Development in Beijing Union University.

References

1. A. S. Ben-Musa, S. K. Singh, and P. Agrawal. Object detection and recognition in cluttered scene using harris corner detection. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies*, pages 181–184, July 2014.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

2. N. Dey, P. Nandi, N. Barman, D. Das, and S. Chakraborty. A comparative study between moravec and harris corner detection of noisy images using adaptive wavelet thresholding technique. *Computer Science*, 2012.
3. O. Haggui, C. Tadonki, L. Lacassagne, F. Sayadi, and B. Ouni. Harris corner detection on a numa manycore. *Future Generation Computer Systems*, 2018.
4. X. Han, M. Ge, and Z. Qinglei. Harris corner detection algorithm on opencl architecture. *Computer science*, 41(7):306–309,321, 2014.
5. C. Harris. A combined corner and edge detector. *The Proceedings of the 4th Alvey Vision Conference*, 1988(3):147–151, 1988.
6. B. A. Jasani, S. Lam, P. K. Meher, and M. Wu. Threshold-guided design and optimization for harris corner detector architecture. *IEEE Transactions on Circuits and Systems for Video Technology*, 2017.
7. D. Li and Y. Tian. Global and local metric learning via eigenvectors. *Knowledge-Based Systems*, 116:152 – 162, 2017.
8. D. G. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the 7th IEEE International Conference on Computer Vision*, page 1150, 2002.
9. S. Mittal and J. S. Vetter. A survey of cpu-gpu heterogeneous computing techniques. *Acm Computing Surveys*, 47(4):1–35, 2015.
10. L. Niu, R. Zhou, Y. Tian, Z. Qi, and P. Zhang. Nonsmooth penalized clustering via ell_p regularized sparse regression. *IEEE Transactions on Cybernetics*, 47(6):1423–1433, June 2017.
11. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.
12. J. Paul, W. Stechele, M. Kröhnert, T. Asfour, B. Oechslein, C. Erhardt, J. Schedel, D. Lohmann, and W. Schröder-Preikschat. Resource-aware harris corner detection based on adaptive pruning. In *Architecture of Computing Systems – ARCS 2014*, pages 1–12, 2014.
13. R. Phull, P. Mainali, Q. Yang, P. R. Alface, and H. Sips. Low complexity corner detector using cuda for multimedia application. In *International Conferences on Advances in Multimedia, MMEDIA*, 2011.
14. K. Pulli, A. Baksheev, K. Kornyakov, and V. Eruhimov. Real-time computer vision with opencv. *Communications of the Acm*, 55(6):61–69, 2012.
15. Z. Qi, F. Meng, Y. Tian, L. Niu, Y. Shi, and P. Zhang. Adaboost-llp: A boosting method for learning with label proportions. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–12, 2018.
16. T. Saidani, L. Lacassagne, J. Falcou, C. Tadonki, and S. Bouaziz. Parallelization schemes for memory optimization on the cell processor: A case study on the harris corner detector. *Transactions on High-Performance Embedded Architectures and Compilers III*, 3:177–200, 2011.
17. V. H. Schulz, F. G. Bombardelli, and E. Todt. A harris corner detector implementation in soc-fpga for visual slam. In *Latin American Robotics Symposium*, pages 57–71. Springer, 2016.
18. J. E. Stone, D. Gohara, and G. Shi. Opencl: A parallel programming standard for heterogeneous computing systems. *Computing in Science and Engineering*, 12(3):66–73, 2010.
19. J. Tang and Y. Tian. A multi-kernel framework with nonparallel support vector machine. *Neurocomputing*, 266:226 – 238, 2017.
20. J. Tang, Y. Tian, P. Zhang, and X. Liu. Multiview privileged support vector machines. *IEEE Transactions on Neural Networks and Learning Systems*, PP(99):1–15, 2017.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34

21. Y. Tian, X. Ju, Z. Qi, and Y. Shi. Improved twin support vector machine. *Science China Mathematics*, 57(2):417–432, Feb 2014.
22. Y. Tian, Z. Qi, X. Ju, Y., and X. Liu. Nonparallel support vector machines for pattern classification. *IEEE Transactions on Cybernetics*, 44(7):1067–1079, 2014.
23. V. D. Weijer, T. Gevers, and J. M. Geusebroek. Edge and corner detection by photometric quasi-invariants. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(4):625–630, 2005.
24. D. Xu, J. Wu, D. Li, Y. Tian, X. Zhu, and X. Wu. Sale: Self-adaptive lsh encoding for multi-instance learning. *Pattern Recognition*, 71:460 – 482, 2017.
25. J. Zhu and K. Yang. Fast harris corner detection algorithm based on image compression and block. In *IEEE 2011 10th International Conference on Electronic Measurement Instruments*, volume 3, pages 143–146, Aug 2011.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93701-4_34