ORB-SLAM accelerated on heterogeneous parallel architectures

Ayoub Mamri[1*], *Mohamed* Abouzahir[1], *Mustapha* Ramzi[1], and *Rachid* Latif[2]

[1] Laboratory of Systems Analysis, Information Processing and Industrial Management, Higher School of Technology of Sale, Mohamed V University of Rabat, Morocco

[2] Laboratory of Systems Engineering and Information Technology, National School of Applied Sciences, Ibn Zohr University of Agadir, Morocco

Abstract. SLAM algorithm permits the robot to cartography the desired environment while positioning it in space. It is a more efficient system and more accredited by autonomous vehicle navigation and robotic application in the ongoing research. Except it did not adopt any complete end-to-end hardware implementation yet. Our work aims to a hardware/software optimization of an expensive computational time functional block of monocular ORB-SLAM2. Through this, we attempt to implement the proposed optimization in FPGA-based heterogeneous embedded architecture that shows attractive results. Toward this, we adopt a comparative study with other heterogeneous architecture including powerful embedded GPGPU (NVIDIA Tegra TX1) and high-end GPU (NVIDIA GeForce 920MX). The implementation is achieved using high-level synthesis-based OpenCL for FPGA and CUDA for NVIDIA targeted boards.

1 Introduction

In most cases, the compute-intensive tasks are managed by CPU, it might be beneficial for the power consumption but the notion of the execution time could be missed. Rather, GPU is usually used for this purpose, specially for graphic processing tasks even though it enforces some limitations on accelerated algorithms, that limitations must be realized in order to acquire an effective gain. Field Programmable Gate Array (FPGA) [1] is proposed as a high-performance scalable compute accelerator in order to benefit from its recommended advantages (improved performance, low cost, reduced energy consumption, more flexible and reliable in different applications), which allows to achieve a high speed and remarkable performance gains. FPGA contains very developed resources in the form of an array of programmable logic blocks, such as Digital-signal-Processing (DSP) that has the capability of Multiply-accumulate (MAC) operation in single instruction cycle, Look-Up Table (LUT) and embedded memory type SRAM. Those resources are designing a modern embedded architecture and often used to implement complex algorithms that make FPGA more attractive choice.

As known FPGA requires a hardware description language such as VHDL or Verilog to handle. In the case of complex algorithms, VHDL and Verilog are often more difficult and unacceptable to most software developers: wherefore High-Level Synthesis (HLS) [2] used for making this task easier. The HLS purpose a hardware description through converting a high-level language based on C/C++ programming language to a hardware model for FPGA taking into consideration the low usage of FPGA resources.

OpenCL [3] is a portable and high-level language framework that provides software developers a powerful capability using multiple embedded devices (such as FPGAs, GPUs, DSPs, and others), moreover, offers an opportunity for accelerating complex algorithms by porting compute-intensive parts into those heterogeneous platforms. It has to be noted there is some difference on the usage of OpenCL between GPU and FPGA, whereas OpenCL for FPGA is a critical challenge compared to OpenCL for GPU, it takes advantages of HLS considered as hardware programming languages that require a deep understanding of FPGA architecture, FPGA on-chip resources characteristics, and to special optimizations. Concerning GPUs there is specific powerful General-Purpose Computing on Graphics Processing Units (GPGPU) developed by NVIDIA with a parallel computing language CUDA [4].

SLAM (Simultaneous Localization and Mapping) [5] is one of the accredited algorithms by autonomous navigation and robotic applications in the ongoing research framework, except that it did not benefit from a complete hardware architectural implementation yet. Besides, they contain compute-intensive parts that need an embedded architecture that allows hardware and software optimization for an efficient and scalable implementation. Toward this goal, researchers aim for heterogeneous architecture whereby the sequential processing parts are loaded by CPU, while accelerators (FPGAs, GPUs) handle the compute-intensive parts for efficient performance and effective speed-up.

[*] Ayoub Mamri: ayoub_mamri@um5.ac.ma

Our contributions: The heterogeneous frameworks open the opportunity to autonomous navigation and robotic applications for discovering new efficient usage of some scalable accelerators and making them a strong tendency to embedding a complex algorithm as SLAM as well in real-time. Toward this end, we attempt to propose a hardware/software optimization of a monocular visual SLAM functional block and Visual Odometry (VO) [6] part as well to reduce its computational time. The detailed discussion about VO algorithm is beyond the scope of this paper.

The purpose of contribution is to present an efficient implementation of a time-consuming functional block. The main contributions are:

- Our work aims to target different heterogeneous architectures CPU-GPU and CPU-FPGA
- A parallel implementation is achieved using OpenCL and CUDA to optimize the most time-consuming functional blocks,
- The target functional block is not obvious to parallelize and require special algorithm modification. Thus, we propose some approaches to deal with sequential algorithm.

The paper is organized as follows: In section II, we present some related work. Section III we paved the way to the targeted functional block. In Section IV, we describe the proposed implementation. In Section V, we give a comparative study between the targeted heterogeneous platforms.

2 Related work

The next SLAM algorithms are notoriously difficult to be implemented in an embedded architecture. In recent years, almost works have been made in research framework are focused on finding a suitable architecture supports a complete end-to-end embedded SLAM system operate in real-time. Hence, researchers have been pushed towards heterogeneous architectures to benefit from powerful devices advantages as DSP, GPU and FPGA. The choice depends on a hardware/software co-design study that provides an overview of the algorithm and the suitable embedded architecture. Meanwhile, heterogeneous architecture implementation and massive parallelism get trends towards high performance scalable compute FPGA accelerator due to low cost and low power needs.

Recent work by abouzahir [7] provides a heterogeneous implementation that aims for low-power embedded architecture CPU-FPGA to implement a VSLAM algorithm. They worked on improving FastSLAM1.0 to a new version FastSLAM2.0 due to an optimization of all blocks, adopted on parallel implementation on FPGA and GPU except image processing part used the FAST detector which is implemented on one-core of CPU using machine learning optimization, moreover, they evaluated few implementations of SLAM algorithms on high-performance machines. As a result, they demonstrated the embedded FPGA accelerators provide significant improvement of SLAM system than GPU accelerators in terms of processing time. Except FastSLAM suffers from impoverishing samples and depleting particles problems, therefore, is not a good choice as SLAM algorithm for large-scale outdoor/indoor environment.

Authors in [8] developed a novel original feature-based stereo VSLAM framework named HOOFR SLAM based on an enhanced bio-inspired feature extractor Hessian ORB - Overlapped FREAK (HOOFR), which is a combination of FAST detector including Hessian score and amended FREAK bio-inspired descriptor. Moreover, they attained to implement it on heterogeneous architecture CPU-GPU. Through this, the Front-End (feature extractor algorithm) part took many advanced strategies to be implemented: First, they ported HOOFR extractor into CPU to exploit all the computing cores using OpenMP, which is more suitable and accelerated than GPU due to machine learning optimization. Second, they ported the Features Matching block into GPU for hardware acceleration due to its computational cost. On the other hand, the Back-End (heart of SLAM) part is improved using a proposal method called “windowed filtering” adapted to the scan matching process instead of a high-cost Bundle Adjustment (BA). Their main result is based on a competitive study that achieves a remarkable gain and effective speed-up; more reliable reconstructed trajectory in some cases and lower cost than stereo ORB-SLAM [9], except that they didn’t adopt a heterogeneous implementation CPU-FPGA that guarantees a low power consumption beside low cost and high performance.

[10] proposed an attractive integrated computing platform that deals with compute-intensive tasks named Heterogeneous Extensible Robot Open (HERO) composed of an Intel Core i5 CPU as host and a high-performance scalable compute FPGA accelerator Intel Arria 10 as device. This platform is developed under needful objects to facilitate research that deals with: heterogeneous computing, algorithm acceleration, compute-intensive component. Within the scope of this paper, they proposed a heterogeneous implementation of HectorSLAM algorithm, whereas they work on accelerating the scan matching process of HectorSLAM on HERO platform, which achieved a significant improvement and remarkable gain, while performing 4-times faster than software implementation with Intel Core I5 CPU and 3-times speed-up against HDL implementation with only Arria 10 SoC.

In contrast, the ultimate goal of this paper is improving the performance of our targeted VSLAM algorithm in terms of computational cost. For this purpose, our hardware-software co-design study adopts aforementioned works [8, 10], where they provide appealing results tackling the computational complexity of SLAM system, furthermore, they worked on heterogeneous systems porting compute-intensive parts of SLAM system precisely the scan matching process into accelerators: The first [8] paves the way to us toward our targeted ORB-SLAM algorithm that shows reliable results than their proposal, except we will be satisfied with the monocular version. The second [10] leads us toward an efficient trend of heterogenous architectures CPU-FPGA.

3 Algorithm description

In this section, we assume steps to pave the way to the optimized functional block, giving an overview of the chosen algorithm and performance evaluation of the system.

3.1 ORB SLAM overview

The selected algorithm is a monocular ORB- SLAM [11], one of the purest Visual SLAM frameworks that operates in real-time, in small and large environments. As shown in figure 1, the system consists of three concurrent threads: tracking, local mapping, loop closing.

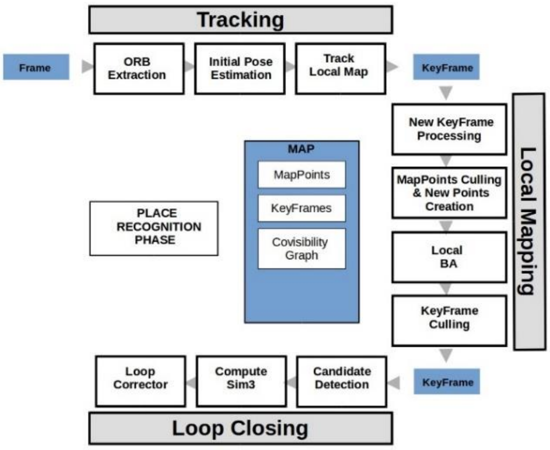

Fig. 1. ORB-SLAM overview.

The tracking thread deals with the camera localization at each frame reception and decides to add it or not to the system. It performs a matching between the previous frame and the current frame and calculates the camera position by an evolution model. In the case where the tracking is lost, the place recognition phase is launched to achieve a global relocalization. If the tracking is successful besides a first estimate of the camera position and a set of matched keypoints, a local map is constructed using the covisibility graph. A second matching phase is performed to identify landmarks in the local map using a projection procedure, then the position of the camera is optimized with the matched keypoints. Finally, the tracking thread decides whether to save or to abandon the keyframe for the next thread.

The local mapping thread processes the keyframes acquired by the tracking thread and execute Local BA to achieve optimal map reconstruction. A matching phase is performed to look for matches in the keyframe connected in the covisibility graph to allow their triangulation. After the initialization of the new points, a selection procedure is realized to keep only the high-quality points based on certain information collected by the tracking thread.

The loop closing thread looks for potential loops in every acquired keyframe. The detection of loop closing leads to calculate the similarity transformation that gives information on the degree of drift accumulated in the loop. Then, the two loops are aligned and the duplicate points are merged. In the end, an optimization of the position graph is performed to achieve global consistency.

3.2 Functional block choice

The performance of the ORB SLAM System is evaluated with the CPU of the targeted platforms (detailed in Section V): Core Intel i5 of laptop machine and ARM Cortex A57 of embedded NVIDIA TX1 board. It has to be noted that the processing time depends on several parameters of algorithm and platform, therefore, we adopt Ubuntu 16.04 version as an operating system in the targeted platforms and besides, we evaluate the system on TUM1 Dataset [12] with monocular images. Table 1 shows the mean of processing time of every functional block (FBs). Among the FBs, Map Point Culling / Creation new map points and Local BA blocks are notoriously time-consuming blocks except they require a dependency with other blocks and a very complicated calculation. Therefore, we selected Initial Pose Estimation which has the third-highest running time and performs 44% of the tracking thread.

Table 1. ORB-SLAM performance evaluation.

	FBs	**Laptop Intel**	**Embedded TX1**
Tracking	ORB Exctraction	19.58	62.32
	Intital Pose Estimation	26.03	67.46
	Track Local Map / KeyFrame Decision	6.42	22.15
	Total (ms)	**52.03**	**151.93**
Local Mapping	New KeyFrame Processing	16.87	42.10
	MapPoints Culling / New Points Creation	103.31	297.45
	Local BA	156.45	469.53
	KeyFrame Culling	6.97	18.51
	Total (ms)	**283.60**	**827.59**
Loop Closing	- Candidate Detection - Compute Sim3 - Loop Corrector ***(ms)***	3.80	5.67

3.3 Map initialization

Map initialization is a part of the second system functional block, which perform 67% of FB2 and 30% of tracking thread. It handles the relative camera pose process between two frames basing on two geometrical models; a fundamental matrix [13] for non-planar scene and a homography [13] for planar scene, to triangulate initial points of the map. Thus, a heuristic method (i.e. ratio of scores) is calculated to select the appropriate geometrical model applied for the current scene whereby an initial reconstruction is achieved. More detailed clarifications in [13].

3.4 Towards optimization

The proposed optimization aims to parallelize the third part of Map initialization with the targeted heterogeneous platforms. Algorithm 1 provides insight into geometrical models M computation (F for the fundamental matrix, H for the homography) inside RANSAC [14] iterations using normalized eight-point and DLT algorithms, as detailed in [13]. For the sake of improved results accuracy of those algorithms, the normalization method has to be carried out before.

Algorithm 2 Geometrical model M computation inside RANSAC iterations it = 200

```
1)  Normalize the detected keypoints.
2)  Perform all RANSAC iterations it for each model M
    and save the solution with highest score:
    a.  Select random points applying 8-point algorithm.
    b.  DLT algorithm to Compute the model M matrix.
    c.  Denormalization.
    d.  Compute current score.
    e.  Test score.
```

Practically, the models M are computed in parallel using C++ multi-threading API (used std::thread class defined in header <thread>). Moreover, normalize function (Norm) is carried out in both H and F. Meanwhile, Norm is called consecutively twice for each M (for current image, reference image) and it handles 2001-2010 keypoints experimentally. Thus, our main idea is to introduce a first step of FB2 optimization toward heterogeneous systems accelerating Norm and reducing memory resources usage. Toward this end, we propose one execution of Norm handling current image and reference image as arguments simultaneously. However, Norm is not parallel in nature, wherefore, we propose special modification to bridge this gap.

4 Towards heterogeneous implementation

In this section, we describe Normalize kernel step by step. Toward this, we developed two versions of normalize function: OpenCL for FPGA and CUDA version for GPU. In the following, we based on OpenCL for FPGA to describe the proposed implementation while the CUDA version could be inferred easily.

4.1 OpenCL for FPGA platform

In OpenCL terminology, the host is always the CPU whereas, FPGA called the device. The host CPU gives the order to the FPGA to execute the calculation. The code executed by FPGA named kernel. The OpenCL architecture provides NDRange, that composed of work-groups which are associable, these work-groups are constituted by work-items, the work-items are active elements in the execution step. Each work-group has a 1D, 2D or 3D identifier in the NDRange, in which the work-item has also a 1D,2D and 3D identifier within the work-group. The data buffering is achieved between host and FPGA memories via PCI-express bus. OpenCL provides fourth types of memory for FPGA with specific usage: global memory that guarantees the data transfer sequentially, constant memory that has the shortest latency, local memory shares data between work-items in the same work-group with low latency, and private memory the fastest memory access, which is dedicated to each work-item work.

4.2 Normalize: accelerated version

Normalize function is computed in the two consecutive frames: reference frame $u_i^r(x_i^r, y_i^r) \in F_r$ and current frame $u_i^c(x_i^c, y_i^c) \in F_c$

$$\bar{\boldsymbol{x}}^{r,c} = \frac{1}{N_{r,c}}\sum_{i=0}^{i=N} x_i^{r,c} \quad , \quad \bar{\boldsymbol{y}}^{r,c} = \frac{1}{N_{r,c}}\sum_{i=0}^{i=N} y_i^{r,c} \qquad (1)$$

$$\begin{aligned}\bar{\mathbf{x}}_{dev}^{r,c} &= \sum_{i=0}^{i=N}\left|\left(x_i^{r,c} - \bar{\mathbf{x}}^{r,c}\right)\right| \\ \bar{\mathbf{y}}_{dev}^{r,c} &= \sum_{i=0}^{i=N}\left|\left(y_i^{r,c} - \bar{\mathbf{y}}^{r,c}\right)\right|\end{aligned} \qquad (2)$$

with $\bar{x}^{r,c}, \bar{y}^{r,c}$: respectively the mean of x and y corner coordinates, $N_{r,c}$: the number of detected corners in both reference and current frame.
The normalized points are givens by the following function:

$$\begin{aligned}x_{i,n}^{r,c} &= \left(x_i^{r,c} - \bar{\mathbf{x}}^{r,c}\right) s_x^{r,c} \\ y_{i,n}^{r,c} &= \left(y_i^{r,c} - \bar{\mathbf{y}}^{r,c}\right) s_y^{r,c}\end{aligned} \qquad (3)$$

With:

$$s_x^{r,c} = \frac{1}{\bar{\mathbf{x}}_{dev}^{r,c}} \quad , \quad s_y^{r,c} = \frac{1}{\bar{\mathbf{y}}_{dev}^{r,c}} \qquad (4)$$

The normalized matrix is given by:

$$T_{r,c} = \begin{pmatrix} s_x^{r,c} & 0 & -\bar{\mathbf{x}}^{r,c} s_x^{r,c} \\ 0 & s_y^{r,c} & -\bar{\mathbf{y}}^{r,c} s_y^{r,c} \\ 0 & 0 & 1 \end{pmatrix} \qquad (5)$$

These equations contain parts that are parallel in nature and other parts that are not obvious to parallelize and require special modifications to adjust to the FPGA kernel. Thus, we propose a new parallel version, see figure 2, including approaches to deal with sequential parts, and NDRange Kernel optimizations [15] to improve data processing and memory access efficiently.

4.2.1 NDRange kernel optimizations

NDRange kernel optimizations are a set of optimizations offered by Altera SDK for OpenCL [16] dedicated for FPGA kernel, we adjust the following to optimize our proposed kernel.

- ***Kernel vectorization (SIMD)***

We used num_simd_work_items attribute for utilizing the global memory bandwidth efficiently by allowing

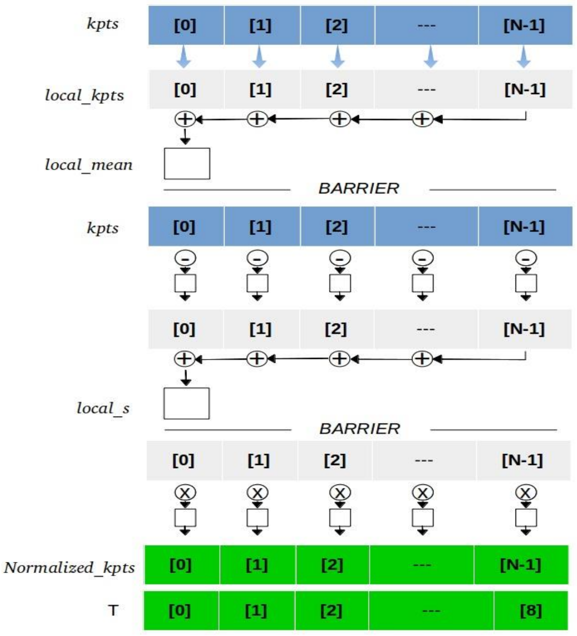

Fig. 2. Data flow of the kernel with OpenCL for FPGA: Constant memory (blue), Local memory (gray), Global memory (green).

multiple work-items to execute in a Single Instruction Multiple Data (SIMD) to achieve higher throughput.

- ***Constant memory (CM)***

We used global _const arguments to handle the buffers kpts (reference image keypoints, current image keypoints) transferred from the host with low latency instead of global memory.

- ***Packetization***

Packetization is a type of vectorization that we used it to packetize kpts in single float2 vector for optimal widening of loads/stores

- ***Local memory (LM)***

We used _local for preloading the packetized buffers kpts to apply the following approach and to share data between work-items in the same work-group and reduce the number of global memory accesses.

4.2.2 Accumulation approach

We adopt this strategy to handle the sequential parts of (1) and (2) which require a preloading of the packetized buffers kpts to LM, whereas each work-item within work-group accumulates 2 separated elements by an offset equal to work_group_size divided by 2. Then this operation is repeated dividing offset by 2 in each iteration until the it becomes lower than 1. The work-items within each work-group must be synchronized with CLK_LOCAL_MEM_FENCE barrier to control and manage the memory accesses properly.

5 Implementation and evaluation

The proposed kernel is implemented and evaluated on three heterogeneous architectures.

5.1 Architectures description

5.1.1 DE1-SoC board

DE1-SoC board is low power embedded hardware designed by Altera System-on-Chip (SoC) FPGA, combines an ARM-based hard processor system (HPS) with FPGA Cyclone V. The HPS includes a Dual-Core ARM Cortex A9 MPCore processor running at 800 MHz, 1GB (256Mx32) DDR3 SDRAM, single hard memory controller, and a rich set of peripherals. On the other hand, the chip integrates FPGA that provides four 50MHz clock sources from clock generator, and 64MB (32Mx16) SDRAM, besides, it consists of several embedded resources: 85K logic elements, 87 DSP blocks, 4.450Kbits embedded memory, 6 Fractional PLLs, and single hard memory controllers.

5.1.2 NVIDIA platforms

Our implementation has adopted two CPU-GPU platforms: a laptop machine and an automotive embedded architecture (NVIDIA Tegra TX1). Table 2, shows the targeted NVIDIA platforms specifications.

Table 2. Platform specification.

	Laptop	**TX1**
CPU	8 Intel core i5	4-cores ARM A57 4-cores ARM A53
CPU clock rate	1.9-2.5 GHz	1.3-1.9 GHz
Cache	3 MB	2 MB
RAM	8 GB	4 GB
GPU	256 CUDA core GeForce 920MX	256 CUDA core Maxwell
GPU clock rate	0.98 GHz	1 GHz
Memory clock rate	2505 MHz	13 MHz

The host of the laptop machine provides an Intel Core i5 4300U CPU with 8 cores operate at 1.9~2.5 GHz and offers 8 GB DDR3 memory with 3 MB of cache memory. The laptop machine integrates also a high-end NVIDIA GeForce 920MX GPU with 256 CUDA core (128 per SM) operates at 980MHz, 2GB global memory and 2505 MHz in the clock rate.

On the other hand, the automotive embedded architecture NVIDIA Jetson TX1 dedicated to embedded application with low power, it is equipped with ARM Cortex A57 quad-core running at 1.9 GHz with 4 GB memory and 2 MB of L2 cache. Moreover, the board integrates a modern powerful embedded GPGPU with Maxwell architecture that contains 256 CUDA core (128 per SM) running at 1GHz, 4GB global memory and clock memory rate about 13MHz.

5.2 Evaluation and results

Table 3 shows the evaluation of the proposed implementation on NVIDIA platforms. Where the high-end NVIDIA GeForce 920MX GPU achieves 2-times speedup than Intel Core i5 CPU, whereas TX1 GPU could not tackle the computational time. Although the kernel code was the same for the two platforms without modifications and the platform specifications were almost similar, the difference in term of computational time is too obvious due to the frequent preloading of data, and heavy memory clock rate for Tegra TX1 GPU.

Table 3. Evaluation of the kernel with two frames and 2048 keypoints.

Margin	**Laptop**		**TX1**	
Device	CPU	GPU	CPU	GPU
Normalize (ms)	0.133	0.073	0.365	0.384

For DE1-SoC board, the proposed kernel with two frames and 2048 keypoints was not supported due to the resource limitation. To overcome this limitation, we have proposed an optimal solution supported by the board which consists of reducing the data size to 1024 and processing a single image. The estimated resources usage of the FPGA required by the proposal use 59% of logic elements, 26% of registers, 65% of memory blocks and 14% of DSP blocks

Table 4 presents the comparative study between the targeted platforms, through an evaluation of the optimal kernel versions.

Table 4. Evaluation of the experimental kernel version with single frame and 1024 keypoints.

	DE-SoC	**TX1**	**Laptop**
Device	**FPGA**	**GPU**	**GPU**
Normalize (ms)	0.160	0.220	0.026

The DE1-SoC FPGA achieved almost an appealing result than Tegra TX1 GPU as an embedded system due to the optimizations applied to the kernel and the memory conception. As is obvious, the compute-intensive parts require a huge preloading of data to memories, thus, the optimizations applied such SIMD and vectorization increase the throughput, besides it helps to reduce the memory accesses providing an efficient usage of the global memory instead the GPU that requires a system to avoid the local memory bank conflict such a Direct Memory Access (DMA). On the other hand, FPGA architecture offers an attractive memory conception that provides a low latency due to the configurable memory layer SRAM that manages data flow in the operational layer.

6 Conclusion

Current works in a SLAM system in the ongoing research aim to tackle with the computational time of the compute intensive parts of the algorithm using heterogeneous NVIDIA platforms such [8, 17]. However, the system requires an optimization to handle the computational complexity. Thus, our work presented a proposed implementation of the algorithm part targeting heterogeneous architectures. Moreover, the main idea was presenting the heterogeneous system FPGA-based capability to tackle with the computational complexity of heavy algorithms by its nature. Thus, a high-performance scalable compute FPGA accelerator as ARRIA 10 is an attractive choice to attain a complete end-to-end embedded SLAM system.

References

1. National instrument, introduction to fpga technology: Top 5 benefits, http://www.ni.com/white-paper/6984/en/. 2012.
2. J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang. High-level synthesis for fpgas: From prototyping to deployment. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 30(4):473–491, April 2011. I
3. Khronos Group, "OpenCL-The open standard for parallel programming of heterogeneous systems,"
4. Li H, Yu D, Kumar A, Tu YC. Performance Modeling in CUDA Streams - A Means for High-Throughput Data Processing, Proc IEEE Int Conf Big Data. pp,301-310,2014.
5. Sebastian Thrun and John J Leonard. Simultaneous localization and mapping. In Springer handbook of robotics, pages 871–889. Springer, 2008.
6. Scaramuzza, D., Fraundorfer, F.: Visual odometry: Part i the first 30 years and fundamentals.
7. Abouzahir, M., Elouardi, A., Latif, R., Bouaziz, S., & Tajer, A. (2018). Embedding SLAM algorithms: Has it come of age? Robotics and Autonomous Systems, 100, 14–26.
8. Nguyen, D.-D., Elouardi, A., Florez, S. A. R., & Bouaziz, S. (2018). HOOFR SLAM System: An Embedded Vision SLAM Algorithm and Its Hardware-Software Mapping-Based Intelligent Vehicles Applications. IEEE Transactions on Intelligent Transportation Systems, 1–16. 8
9. R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," IEEE Trans. Robot., vol. 33, no. 5, pp. 1255–1262, 2017.

10. Shi, X., Cao, L., Wang, D., Liu, L., You, G., Liu, S., & Wang, C. (2018). HERO: Accelerating Autonomous Robotic Tasks with FPGA. 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).
11. Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: A Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics, 31(5), 1147–1163.
12. http://vision.in.tum.de/data/datasets/rgbd-dataset/download
13. R.Hartley and A. Zisserman, Multiple View Geometry in Computer Vision, 2nd ed., Cambridge, U.K. 2004.
14. E. Michaelsen, W. V. Hansen, M. Kirchhof, J. Meidow, and U. Stilla, "Estimating the essential matrix: GOODSAC versus RANSAC," in Proc. Symp. ISPRS Commission III Photogramm. Comput. Vis. (PCV),Germany, Sep. 2006.
15. Jia, Q., & Zhou, H. Tuning Stencil codes in OpenCL for FPGAs. 2016 IEEE 34th International Conference on Computer Design (ICCD).
16. Altera sdk for OpenCL, Best Practices Guide, https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf
17. J. Li, G. Deng, W. Zhang, C. Zhang, F. Wang and Y. Liu. Realization of CUDA-based real-time multi-camera visual SLAM in embedded systems. Journal of Real-Time Image Processing. 2019.