IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

Augmenting Embedded Systems for Computer Vision Applications

Neelanjan Goswami[1]

U.G. Student, Department of Electronics and Communication, Bangalore Institute of Technology, Bengaluru, India[1]

Abstract: Enabling CV, computer vision applications that use low levels of power on embedded devices (called as low-power embedded systems), also poses new sets of problems for developers of embedded applications. These technologies incorporate multiple functionalities, such as deep learning-based image recognition, simultaneous localization and mapping tasks. In order to ensure real-time behaviours and at the same time energy limitations to conserve battery on the mobile network, they are distinguished by rigorous efficiency constraints. While heterogeneous embedded boards are becoming widespread at low power costs for their high computing power, they require a time-consuming customization of the entire application (i.e. mapping application blocks to GPU and CPU processing elements and their synchronization) to efficiently leverage their ability. For such an embedded SW customization, distinct languages and environments have been suggested. However, in complex real situations, they also find limits, since their implementation is mutually exclusive. This paper provides a detailed architecture focused on a heterogeneous parallel programming paradigm that integrates OpenMP, PThreads, OpenVX, OpenCV, and CUDA to better take advantage of various parallel levels while semi-automatic customization is guaranteed. The paper demonstrates how to interface, synchronize, and implement certain languages and API frameworks to customize an ORB - SLAM program for the NVIDIA Jetson TX2 board.

Keywords: low-power embedded system, computer vision, embedded vision, OpenCV, OpenVX, PThreads

I. INTRODUCTION

In current CPS, cyber-physical systems, machine vision is becoming omnipresent. Using computer imaging and intelligent algorithms to decode meaning from images or video streams is the main objective. In the framework of cyber-physical devices, computer vision implementations typically consist of many computer-intensive kernels that introduce multiple functionalities, ranging from image detection to mapping and localization at the same time (SLAM).

It is not an immediate and easy job to create and optimize such software for an embedded device. In addition to functional correctness, developers have to deal with non-functional aspects, such as performance, power usage, energy conservation and real-time constraints. Architecture-dependent optimization encompasses two major dimensions: block-level and system-level. The first one is more simplistic and includes the parallelization or reimplementation of single kernels by particular languages or programming environments such as OpenCL, OpenMP, PThreads or CUDA for the target board accelerators (e.g., GPU, DSP, or multi-cores). The system-level optimization targets the total power usage of the system, memory space, and overhead of inter-process communication. Space discovery mapping involves investigating the various techniques to map each of these kernels to the board's proper processing components and evaluating the resulting effect on the architecture constraints.

One of the fastest in gaining consensus as a programming framework and API library for system - level optimizations in the environment of embedded - vision is OpenVX. Such a platform is designed to optimize the portability of features and performance across numerous hardware systems, offering a computer vision interface that tackles various hardware architectures effectively with limited impact on software applications.

Nonetheless, any real embedded vision program involves the incorporation of OpenVX with user-defined C or C++ code due to the restriction of OpenVX to model complicated functions by data - flow graphs and the incompleteness of the primitive OpenVX library. On the one side, multi-core parallelization strategies will support the user-defined code, thereby creating heterogeneous parallel environments (i.e. multi-core + GPU parallelism). In the other hand, because of OpenVX's private and non-user-controlled memory stack, such an aggregation leads to the sequentialization of multiple execution environments, with a consequent heavy effect on optimization at the device level.

A paradigm for heterogeneous parallel programming of embedded vision systems is provided in this article. This allows numerous programming frameworks, i.e. OpenMP, PThreads, OpenVX, OpenCV, and CUDA, to be merged to better take advantage of varying degrees of parallelism while maintaining semi-automatic customization.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

In order to optimize a current SLAM application for the widespread NVIDIA Jetson TX2 board, the paper provides an overview of the shortcomings encountered by implementing state-of-the-art parallel programming environments. Ultimately, it presents the outcomes of the mapping space exploration we conducted by taking into account output, power consumption, energy efficiency, and architecture constraints of result quality. As follows, the document is structured. An introduction to OpenVX in Part I and the corresponding work is provided in Part II. Part III provides, by a case study, an overview of parallel programming environments for embedded vision applications. The suggested structure is presented in Part IV. Part V discusses the experimental findings, while the closing remarks are dedicated to Part VI.

II. History and Associated Works

A platform is needed for the development and enhancement of embedded vision systems, taking into account numerous architecture limitations (i.e., performance, power consumption) and OpenVX is such a framework (i.e., performance, power consumption). To describe a high level and architecture-independent representation of the program, it relies on a graph - based model. Through the use of a series of primitives, that are provided by the framework and reflect the most widely used features and functionalities and data objects in computer vision algorithms, such as scalars, arrays, matrices and images, as well as high-level data objects such as histograms, image pyramids, and look-up tables, such a representation is modularly created by the user.

Through the use of the libraries of architecture-oriented application of primitives and data-structures offered by the board provider, the high-level representation (i.e. the graph) is then automatically optimized and embedded.

A computer vision algorithm is specified by the creator by defining kernels as nodes and data objects as parameters (see the example in Fig. 1). Each graph node is known as a function kernel that can operate on any of the heterogeneous target board processing units. Indeed, the application graph reflects the division into blocks of the whole application, which can be performed through multiple hardware accelerators (e.g., CPU cores, GPUs, DSPs).

The flow of programming begins with the development of a context for OpenVX to handle references to all objects used. The code creates the graph on the basis of this context and produces all the data objects needed. Then as graph nodes, it instantiates the kernel and produces their relations. The layout first tests the consistency and validity of the graph (e.g., checking the coherence of the data type across nodes and the lack of cycles) and, eventually, processes the graph. It releases all the generated data objects, the graph, and the context at the end of the code execution.

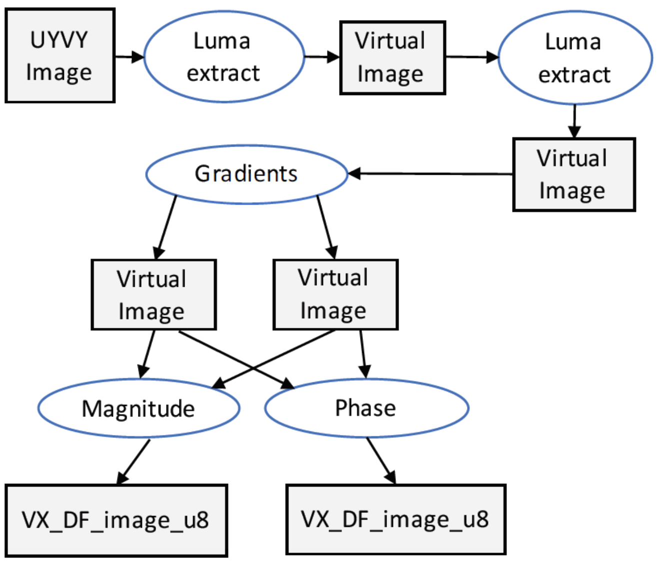


Fig. 1 A graphical representation of a sample OpenVX application

In Fig 1, input is taken from a blurred input file and the application computes the gradient magnitude and gradient phase. The nodes of magnitude and step are computed separately, so that one does not depend on the other's output. OpenVX does not enable them to be executed concurrently or in parallel, so it requires the board vendor's runtime manager to settle on the mapping and execution plan.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

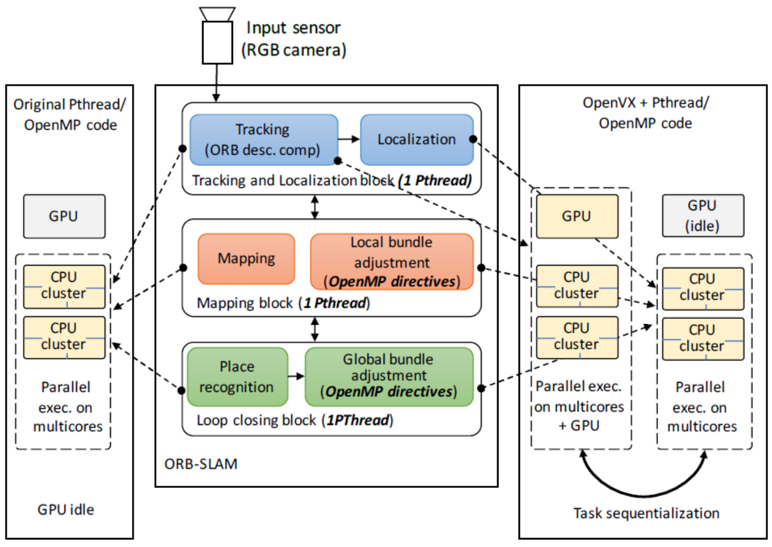
OpenVX allows various mapping techniques between nodes and processing elements of the heterogeneous board to be implemented by implementing any vendor library that implements the graph nodes as primitives of Computer Vision, by targeting different de-sign restrictions (e.g., performance, power, energy efficiency).

In order to analyse the use of OpenVX for embedded vision numerous studies have been presented. The authors present a new version of OpenVX targeting CPUs and GPU-based devices by exploiting numerous techniques for analytical optimization. By evaluating three different OpenVX optimizations, the authors investigate how OpenVX responds to different data access patterns: kernel merging, data tiling, and parallelization through OpenMP. The authors implement ADRENALINE in, a novel method for rapid prototyping and optimization of OpenVX frameworks for multi-core accelerator heterogeneous SoCs. We suggested a methodology for incorporating a model-based design environment with OpenVX. For model-based architecture, parametrization, and validation of computer vision systems, the technique enables the use of MATLAB or Simulink. Then for hardware and constraints-aware device tuning, it facilitates the automatic synthesis of the application model into an OpenVX definition.

III. STUDY OF PARALLEL PROGRAMMING FOR EMBEDDED VISION VIA THE ORB-SLAM CASE STUDY

To explain the constraints of the state - of - the - art environments for embedded vision implementations for parallel programming and the contribution of the proposed framework, we first present the case study, which will be included in the following sections as a model. The case study, ORB-SLAM [1], is a standard genuine embedded application that is used in numerous ways, from automobile to robotic systems. The target platform is the NVIDIA Jetson TX2, which is a generalized and low-cost embedded module.

When RGB camera sensors are adopted, ORB-SLAM solves the simultaneous localization and map-ping problem. In a wide range of settings, it computes, in real-time, the camera trajectory and a sparse 3D recreation of the scene, ranging from brief hand-held sequences of a desk to a vehicle driven through multiple blocks of the area. This generates a 3D map from an input stream and/or performs localization. By considering the actual map. Three key blocks comprise the application as shown in Fig 2 below:



Input sensor (RGB camera)
Original Pthread/ OpenMP code
GPU
CPU cluster
CPU cluster
Parallel exec. on multicores
GPU idle
Tracking (ORB desc. comp)
Localization
Tracking and Localization block (1 Pthread)
Mapping
Local bundle adjustment (OpenMP directives)
Mapping block (1 Pthread)
Place recognition
Global bundle adjustment (OpenMP directives)
Loop closing block (1PThread)
ORB-SLAM
OpenVX + Pthread/ OpenMP code
GPU
GPU (idle)
CPU cluster
CPU cluster
CPU cluster
CPU cluster
Parallel exec. on multicores + GPU
Parallel exec. on multicores
Task sequentialization

(a) (b)

Fig. 2. Overview of program and execution templates for ORB - SLAM: (a) the initial code (parallelized for multicore), (b) the state-of-the-art implementation of OpenVX.

- The monitoring and localization block computes the visual attributes, locates the agent in the environment, and communicates the map update information to the mapping block in the event of major inconsistencies with the map already saved and the input stream. This block efficiency greatly depends on the processing rate i.e., the supported frame rate per second) and the key power usage of the entire program.
- Using information (detected map changes) submitted by the localization block, the mapping block updates the world map. In the case of a well-consolidated map, you should shut down this module to conserve machine energy.
- The loop closing block attempts to change the error of scale drift accumulated during the input analysis, which is necessary when a monocular vision system is implemented (i.e., RGB camera).

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

This block updates the mapped information when a loop is found in the agent pathway via a large latency heavy computation, in which the first two blocks must be halted. This will cause the agent to lose tracking and localization data and as a result, to briefly lose the agent. As a consequence, this block's calculation efficiency (run on demand) is critical for the accuracy of the results of the whole application.

Due to their simultaneous execution model, the three blocks are implemented in the best ORB - SLAM implementation at the state of the art to be executed in parallel by PThreads on shared-memory multiprocessors. Moreover, since the task of package modification, both local in the mapping block and global in the closing loop block, may have long latencies, it is a primary parallelization goal. For directive-based automated parallelization, its nested and data-independent loops apply well. Therefore, for concurrent execution on multi-core, the state – of – the – art code is available with OpenMP directives. For parallel execution on the GPU, no block is initially considered (see Figure 2(a)).

The manual execution of every GPU sub - block is beyond the scope of this job. Instead, we consider the semi - automatic embedding of the application through OpenVX due to the difficulty of such a parallelization task for this application class, but considering different architecture constraints (power consumption and energy efficiency besides performance).

On the contrary, the OpenVX program must also be merged into standard C or C++ code because of the constraint of OpenVX to model complex systems by data-flow graphs and the incompleteness of the vendor repository. In the ORB-SLAM case study, a data-flow graph will model only the monitoring sub-block and is worth optimizing for CPU/GPU execution. Although the majority of the code will also run on multicores, the execution of the two environments (OpenVX + CUDA and the rest) is then sequenced, as stated in Part IV, to allow coordination and synchronization. The purpose of the proposed approach is to incorporate the two environments. There are some benefits of such an integration, such as multi-level parallel deployment of the program and improved mapping space between tasks and elements of processing to be explored.

We depend on common libraries of computer vision functions offered by vendors of the target board (i.e., Vision-Works for NVIDIA boards). The library can be expanded through user-defined or third-party CUDA kernels, which are embedded as custom nodes in the OpenVX implementation.

IV. ENHANCEMENTS USING HETEROGENEOUS PARALLEL PROGRAMMING

The overview of the proposed structure is illustrated in Figure 3. Six languages and parallel programming environments (in the following environments) are considered: C or C++, OpenCV, OpenVX, OpenMP PThreads and CUDA. The heterogeneity of the environment makes it possible to implement various ap-plication blocks in the most suitable style, such as C/C++ for control pieces, PThreads for concurrent CPU execution functions, OpenMP for directive-based automated code chunk parallelization, CUDA for any GPU kernel (if available) amplification, and OpenVX for primitive data flow routine parallelization. OpenCV was selected to introduce modern I/O communication protocols via standard data structures and APIs for computer vision applications. This makes it possible for embedded vision systems to be compact and integrated effectively with some other standard-compliant device.

We consider the ubiquitous and most common NVIDIA Jetson TX2 as the target platform as a running example for the sake of consistency and without lack of generality. Such an embedded board relies on a shared memory architecture in which the unified memory space is shared by two separate CPU clusters (four Cortex-A57 CPU cores and two Denver CPU cores) and a GPU with two symmetric multiprocessors.

The stack layer involved in the parallel execution of each environment is illustrated at the top of Figure 3. It depends on two key sections:

- A user - controlled stack that allows memory-based communication between processes running on various CPUs to be shared. They include OpenMP-generated C/C++ processes, OpenCV APIs, PThreads, and processes.

- A private (not user - controlled) stack that allows communication between OpenVX graph nodes running on separate CPUs or on the GPU and is generated and maintained by OpenVX. The operating system maps the functions relevant to the user-controlled stack to the CPU cores (i.e., Linux Ubuntu for the NVIDIA Jetson). OpenVX functions are mapped by the OpenVX runtime framework to the CPU cores or GPU multiprocessors.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

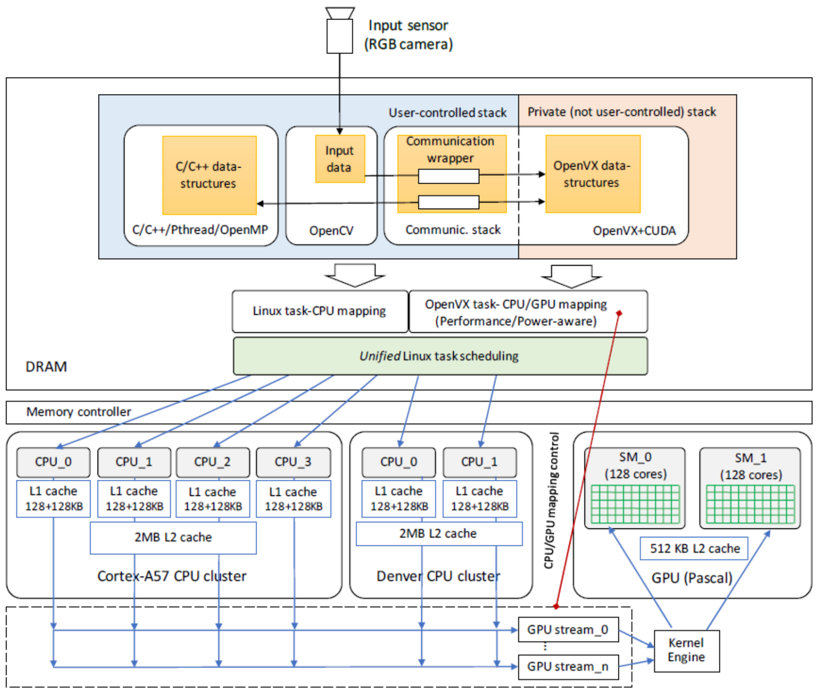


Fig. 3. An overview of Framework: an embedded vision application's task mapping, memory stack and task scheduling layers built on the NVIDIA Jetson TX2 board with the proposed process.

We connect the two parts to a single unified scheduling engine in order to allow the total unification of the two parts, to avoid the sequentialization of the two sets of tasks, and to avoid the consequent overhead synchronization. This helps the operating system to plan all the tasks mapped to the Processor cores (of both stack parts), whereas the OpenVX runtime system manages the GPU work scheduling, CPU - to - GPU connectivity and synchronization (i.e., GPU stream and kernel engine). To do this we are proposing a C or C++ & OpenVX template - based communication wrapper that provides memory access to the OpenVX data structures on the private stack and complete control of the C or C++ environment's OpenVX context execution.

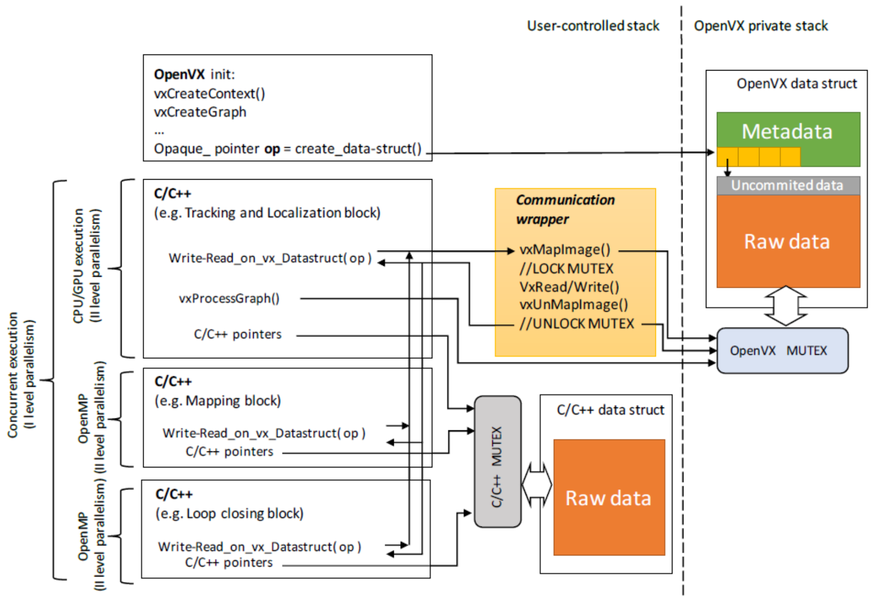


Fig. 4. Integration of the communication wrapper in the system

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

An overview of the wrapper and its integration within the device is shown in Figure 4. The initialization stage of OpenVX creates the background of the graph and allocates the private data structures. This allocation returns invisible pointers to the assigned segments of memory i.e. pointers to private memory areas where the programmer does not know about the structure.

In order to access the private data structures via the opaque pointers, OpenVX read and write primitives (Write - Read - on - vx - Datastructure() in the figure) were specified. The primitives are invoked from the C/C++ context and set up a mutex framework for security access to the OpenVX data structures via the contact wrapper APIs. The same mutex is shared with the overall graph processing OpenVX runtime system (in the figure, vxProcessGraph()). As a result, the protocol maintains compatibility between the OpenVX and C/C++ contexts while operating simultaneously on multicores during access to the shared data structures.

It is worth remembering that the execution of the data-flow driven OpenVX code begins with the invocation of the overall graph processing, which is done in the C/C++ environment. As seen in Figure 4, separate C/C++ threads will simultaneously execute such an invocation, and each invocation requires the mapping and scheduling of the corresponding instance of the graph. Synchronization between the various concurrent OpenVX graph executions and the C/C++ calling environments is facilitated by the proposed contact wrapper and mutex scheme.

When accessing shared data structures, standard mutex frameworks are followed to synchronize all other C/C++-based contexts belonging to the user-controlled stack. The coordination wrapper based on mutex permits multi-level parallel execution of the program. Using the ORB-SLAM case study as an example, the first degree of parallelism is introduced by the PThreads, which run the application's three key modules on separate CPU cores.

Then, the tracking block of the first module is implemented in OpenVX and run on a CPU core and on the GPU. The parallel implementation of the graph nodes offloaded on the GPU is provided by the OpenVX library vendor (i.e., NVIDIA VisionWorks for our case study) and are optimized for the specific GPU architecture. In case two nodes of the OpenVX graph are independent (see the example of Fig. 1), they are executed concurrently.

Finally, OpenMP provides another level of parallelism when a block is enriched with parallel directives (e.g., Mapping and Loop closing blocks in the example). Each of these blocks is executed in parallel by the threads generated automatically by the compiler, which run on the available CPU cores.

V. RESULTS OF THE EXPERIMENT

The system has been added to eventually embed the ORB-SLAM application on the Jetson TX2. We began with the most successful state-of-the-art parallel implementation. We then modularly implemented the various parallel worlds provided by the structure as follows:

- Version 2 (PThreads + OpenMP): Version 1 is extended by allowing parallelism with OpenMP. It explicitly parallelizes the role of package change, both local in the mapping block and global in the closing block of the loop.

- Version 3 (PThreads + OpenVX): Expands version 1 by adding the monitoring sub-block in OpenVX (i.e. with PThreads, without OpenMP parallelism).

- Version 4 (PThreads + OpenMP + OpenVX): Expands version 3 by making OpenMP as well.

- Version 5 (PThreads + OpenVX + CUDA): we have reused a CUDA kernel starting with version 3 that implements the ORB primitive in the tracking sub-block. The related OpenVX VisionWork primitive was modularly replaced with such a more streamlined kernel.

- Version 6 (PThreads + OpenMP + OpenVX + CUDA): Version 5 is extended by supporting OpenMP too.

By using an open-source dataset called KITTI dataset, which is a common and popular benchmark for vision applications, we validated and tested all models. The dataset consists of video streams recorded by traveling in the mid-sized city of Karlsruhe, in rural areas and on highways around a vehicle fitted with an RGB camera. We present the results obtained on sequence number 13 for the sake of space, as it is most important to illustrate the variation of workload in all three ORB-SLAM blocks and the resulting impact on the geometry constraints.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

We have designed the Jetson TX2 board with two separate settings for the evaluation: low frequency (75 percent) and high frequency (100 percent). They reflect the frequency setting of the four components of the board i.e. the four cores of the Cortex - A57 cluster (mean and maximum frequency 1.42 GHz and 2.035 GHz respectively), the two cores of the Denver cluster (1.42 GHz and 2.035 GHz), the GPU (1.032 GHz and 1.3 GHz), and the memory (1.062 GHz and 1.866 GHz).

Version 6 is a case where converting to the parallelism of OpenMP does not have improved performance than the version of Pthread + OpenVX + CUDA thus increasing peak power consumption. In the highest frequency setting, version 6, on the other hand gives improved QoS. This is due to the fact that OpenMP is exclusively concerned with the phases of bundle change, which affect the missed frame while not impacting the assisted FPS. In the medium frequency setting, this does not happen since the CPU frequency at which such a kernel is run does not allow the monitoring block to conform with the real time constraints. We find that version 3 is the most energy efficient for the medium frequency setup and offers the best results for QoS. The best output is given by version 5 and does not require the worst power consumption. Version 6 promises the best output, and with the highest power consumption, it pays almost the best QoS (99.8 percent). Version 5 provides the best trade-off in terms of output and power usage for the highest frequency setting, while version 6 provides the best trade-off in terms of performance, energy efficiency, and QoS.

Finally, the experimental findings demonstrate how the various models and the frequency setup of the single processing modules for each of them provide a very broad mapping area to be explored (which is out of the scope of this work). Such a room will provide the best option for any of the design constraints considered, such as performance, power usage, energy efficiency, and service quality.

The results for average and maximum frequency settings are displayed in Tables I and II, respectively. The best performances are listed in bold. The mapping columns list the number of computing elements used during calculation by the various models. The PThreads, by having one core per block, ensure the minimum degree of parallelism. OpenMP was set to use the highest number of CPU cores available (6). Just OpenVX/CUDA activates the GPU.
The FPS and Time per Frame columns report device output information and in fact, FPS is the maximum number of frames per second allowed by the embedded system. The columns emphasize how the ultimate output is affected by each degree of parallelism. The tables detail the overall energy consumed for the measurement of the entire stream, the average and peak capacity, and the average energy per frame, to explain the impact of the various versions on power and energy efficiency.

TABLE 1. KITTI - FPS AND TIME PER FRAME VALUES, SEQUENCE 13, 75 % OF THE FREQUENCIES

Version	**Mapping**									
	A57	**Denver**	**GPU SM**	**FPS**	**Time per frame (ms)**	**Energy (J)**	**Avg Power (W)**	**Peak Power (W)**	**% frame processed**	**Energy per frame (J)**
Version 1	3	-	-	13.2	13.1	1,203	3.34	4.05	3,097/3,281 (94.4%)	0.357
Version 2	4	2	-	13.9	12.4	1,123	3.44	5.24	3,022/3,281 (92.1%)	0.373
Version 3	3	-	2	19.3	18.3	1,034	3.73	5.82	3,280/3,281 (99.9%)	0.378
Version 4	4	2	2	19.3	19.6	1,252	5.75	7.85	3,261/3,281 (99.4%)	0.383
Version 5	3	-	2	23.4	23.2	1,185	3.67	5.66	3,269/3,281 (99.0%)	0.363
Version 6	4	2	2	23.4	23.2	1,197	5.75	7.92	3,272/3,281 (99.8%)	0.367

TABLE 2. KITTI - FPS AND TIME PER FRAME VALUES, SEQUENCE 13, 100 % OF THE FREQUENCIES

Version	**Mapping**									
	A57	**Denver**	**GPU SM**	**FPS**	**Time per frame (ms)**	**Energy (J)**	**Avg Power (W)**	**Peak Power (W)**	**% frame processed**	**Energy per frame (J)**
Version 1	3	-	-	16.8	59.1	1,917	5.85	8.55	3,264/3,281 (99.5%)	0.586
Version 2	4	2	-	17.2	57.8	1,968	6.01	9.62	3,268/3,281 (99.6%)	0.602
Version 3	3	-	2	21.3	46.2	1,956	5.96	9.53	3,274/3,281 (99.8%)	0.595
Version 4	4	2	2	21.2	46.1	1,943	7.99	11.02	3,281/3,281 (100%)	0.592
Version 5	3	-	2	29.4	34.2	1,897	5.94	9.66	3,275/3,281 (99.0%)	0.580
Version 6	4	2	2	29.3	34.4	1,843	7.69	11.70	3,280/3,281 (99.9%)	0.563

The tables stress that the output (FPS) given by the various versions is as predicted, strictly associated with power consumption. Via the various degrees of parallelism, allowing all the processing elements contributes to the maximum output at the expense of higher peak capacity. We find, however that OpenMP does not make performance improvements in the overall heterogeneous sense in all situations.

Finally, information about quality of service (QoS) outcomes is recorded in the tables. It contains the number of frames correctly processed for the overloading of the processing elements against those missed. Frame skipping is triggered by the mapping and loop closing blocks that operate the calculation of the package change and their delay stops the monitoring block from evaluating new frames due to job overload. A specification restriction is the highest number of frames missed tolerated as it requires the application's QoS such as the number of times the device is missing (see Part III).

VI. Conclusion

A paradigm for heterogeneous parallel programming of embedded vision systems was introduced in this article. The paper first provided an overview of the actual shortcomings of the most important and used environments as applied singularly for parallel computing embedded vision implementations at the state of the art. The paper then explained how the framework enables certain environments, i.e. OpenCV, OpenMP, OpenVX, PThreads and CUDA, to be combined to better leverage various degrees of parallelism while semi-automatic customization is assured. Via a real case study, i.e., an ORB - SLAM program, which was adapted for an embedded NVIDIA Jetson TX2 surface, the paper introduced the review and structure. Eventually, the paper showed that we achieved substantially better results over various design constraints, i.e. output, power consumption, energy usage, and result quality thanks to the greater mapping space produced and the multi-level parallelism offered by such heterogeneous parallel programming.

ACKNOWLEDGMENT

With outstanding direction from my professor, **Dr. A. B. Kalpana**, the paper as well as the job behind it would not have been feasible already. From the first to the final draft of this article, the excitement, expertise and exact emphasis on fine details were a tremendous inspiration and kept my work on track. **Dr. Sree Ranga Raju M. N.**, my professor at the Bangalore Institute of Technology, also looked through my transcriptions and answered several questions about the terminology and meaning of the hardware definition with unfailing patience. I am also grateful for the insightful feedback offered at my college's research block by the anonymous peer reviewers. This thesis has been enriched in many ways by the generosity and knowledge of all and everyone and saved me from many errors; those that certainly remain are my own fault alone.

References

[1]. R. Mur-Artal, J. M. M. Montiel, and J. D. Tards, “Orb-slam: A versatile and accurate monocular slam system,” IEEE Transactions on Robotics, vol. 31, no. 5, pp. 1147–1163, Oct 2015.
[2]. B. Meus, T. Kryjak, and M. Gorgon, “Embedded vision system for pedestrian detection based on hog+svm and use of motion information implemented in zynq heterogeneous device,” vol. 2017-September, 2017, pp. 406–411.
[3]. G. Tagliavini, G. Haugou, A. Marongiu, and L. Benini, “Adrenaline: An openvx environment to optimize embedded vision applications on many-core accelerators,” in International Symposium on Embedded Multicore/Many-core Systems-on-Chip, 2015, pp. 289–296.
[4]. N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” vol. 1. IEEE, 2005, pp. 886–893.
[5]. N. Dalal, B. Triggs, and C. Schmid, “Human detection using oriented histograms of flow and appearance,” vol. 3952. Berlin Heidelberg: Springer, 2006, pp. 428–441.
[6]. NVIDIA Inc., “VisionWorks,” https://developer.nvidia.com/embedded/visionworks
[7]. Khronos Group, “OpenVX: Portable, Power-efficient Vision Processing,” https://www.khronos.org/openvx
[8]. G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces,” in 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, Nov 2007, pp. 225–234.
[9]. M. Hahnle, F. Saxen, M. Hisung, U. Brunsmann, and K. Doll, “Fpgabased real-time pedestrian detection on high-resolution images,” in 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, Jun. 2013, pp. 629–635.
[10]. J. Rettkowski, A. Boutros, and D. Göhringer, “Real-time pedestrian detection on a xilinx zynq using the hog algorithm,” in 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), Dec. 2015, pp. 1–8.
[11]. Andargie, F.A., Rose, J., Austin, T., Bertacco, V.: Energy efficient object detection on the mobile gp-gpu. In: 2017 IEEE AFRICON. pp. 945–950 (Sept 2017)
[12]. R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, “DTAM: Dense tracking and mapping in real-time,” in Proc. IEEE Int. Conf. Comput. Vision, Barcelona, Spain, Nov. 2011, pp. 2320–2327.
[13]. S. Lovegrove, A. J. Davison, and J. Ibanez-Guzman, “Accurate visual ´ odometry from a rear parking camera,” in Proc. IEEE Intell. Vehicles Symp., 2011, pp. 788–793.
[14]. Appiah, K., Hunter, A., Dickinson, P., Meng, H.: Implementation and applications of tristate self-organizing maps on fpga. IEEE Transactions on Circuits and Systems for Video Technology 22(8), 1150–1160 (Aug 2012)
[15]. . Dekkiche, D., Vincke, B., Merigot, A.: Investigation and performance analysis of openvx optimizations on computer vision applications. In: Int. Conf. on Control, Automation, Robotics and Vision, pp. 1–6 (2016)

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Vol. 9, Issue 10, October 2020

DOI 10.17148/IJARCCE.2020.91024

[16]. Dunbabin, M., Grinham, A.: Quantifying spatiotemporal greenhouse gas emissions using autonomous surface vehicles. Journal of Field Robotics 34(1), 151–169 (2017)
[17]. Tagliavini, G., Haugou, G., Marongiu, A., Benini, L.: Adrenaline: An openvx environment to optimize embedded vision applications on many-core accelerators. In: Int. Symp. on Embedded Multicore/Many-core Systems-on-Chip, pp. 289–296 (2015)
[18]. Wang, H., Wei, Z., Wang, S., Ow, C.S., Ho, K.T., Feng, B.: A vision-based obstacle detection system for unmanned surface vehicle. In: Int. Conf. on Robotics, Automation and Mechatronics, pp. 364–369 (2011)
[19]. C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in Proc. IEEE Int. Conf. Robot. Autom., Hong Kong, Jun. 2014, pp. 15–22.
[20]. O. D. Faugeras and F. Lustman, "Motion and structure from motion in a piecewise planar environment," Int. J. Pattern Recog. Artif. Intell., vol. 2, no. 03, pp. 485–508, 1988.
[21]. Yang, K., Elliott, G.A., Anderson, J.H.: Analysis for supporting real-time computer vision workloads using openvx on multicore+gpu platforms. In: Int. Conf. on Real Time and Networks Systems, RTNS '15, pp. 77–86 (2015)
[22]. G. Tagliavini, G. Haugou, A. Marongiu, and L. Benini, "ADRENALINE: an OpenVX environment to optimize embedded vision applications on many-core accelerators," in IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), 2015, pp. 289–296.
[23]. H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Scale drift-aware large-scale monocular SLAM," presented at the Proc. Robot.: Sci. Syst., Zaragoza, Spain, Jun. 2010.
[24]. H. Strasdat, A. J. Davison, J. M. M. Montiel, and K. Konolige, "Double window optimisation for constant time visual SLAM," in Proc. IEEE Int. Conf. Comput. Vision, Barcelona, Spain, Nov. 2011, pp. 2352–2359.

BIOGRAPHY

My name is **Neelanjan Goswami** and currently pursuing Bachelors in Engineering, specializing in Electronics and Communication (2017 – 2021). Currently, I am the head of the technical department of the ECE club and having keen interests in smart cyber physical systems and embedded system designing as well. I am currently researching in the interdisciplinary fields that strive to improve the functioning and automation of various process, making human interaction as little as possible in environments where a human life might be under some kind of danger or unable to avoid any danger.