

Performance and energy-efficient implementation of a smart city application on FPGAs

Original

Performance and energy-efficient implementation of a smart city application on FPGAs / Arif, A., Barrigon, F.A., Gregoretti, F., Iqbal, J., Lavagno, L., Lazarescu, M.T., Ma, L., Palomino, M., Segura, J.L.L.. - In: JOURNAL OF REAL-TIME IMAGE PROCESSING. - ISSN 1861-8200. - ELETTRONICO. - 17:(2020), pp. 729-743. [10.1007/s11554-018-0792-x]

Availability:

This version is available at: 11583/2710367 since: 2020-07-02T01:28:25Z

Publisher:

Springer Verlag

Published

DOI:10.1007/s11554-018-0792-x

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

Springer postprint/Author's Accepted Manuscript

This version of the article has been accepted for publication, after peer review (when applicable) and is subject to Springer Nature's AM terms of use, but is not the Version of Record and does not reflect post-acceptance improvements, or any corrections. The Version of Record is available online at: <http://dx.doi.org/10.1007/s11554-018-0792-x>

(Article begins on next page)



Performance and energy-efficient implementation of a smart city application on FPGAs

Arslan Arif¹ · Felipe A. Barrigon² · Francesco Gregoretti¹ · Javed Iqbal¹ · Luciano Lavagno¹ · Mihai Teodor Lazarescu¹ · Liang Ma¹ · Manuel Palomino² · Javier L. L. Segura²

Received: 1 February 2018 / Accepted: 17 May 2018
© The Author(s) 2018

Abstract

The continuous growth of modern cities and the request for better quality of life, coupled with the increased availability of computing resources, lead to an increased attention to smart city services. Smart cities promise to deliver a better life to their inhabitants while simultaneously reducing resource requirements and pollution. They are thus perceived as a key enabler to sustainable growth. Out of many other issues, one of the major concerns for most cities in the world is traffic, which leads to a huge waste of time and energy, and to increased pollution. To optimize traffic in cities, one of the first steps is to get accurate information in real time about the traffic flows in the city. This can be achieved through the application of automated video analytics to the video streams provided by a set of cameras distributed throughout the city. Image sequence processing can be performed both peripherally and centrally. In this paper, we argue that, since centralized processing has several advantages in terms of availability, maintainability and cost, it is a very promising strategy to enable effective traffic management even in large cities. However, the computational costs are enormous, and thus require an energy-efficient High-Performance Computing approach. Field Programmable Gate Arrays (FPGAs) provide comparable computational resources to CPUs and GPUs, yet require much lower amounts of energy per operation (around 6× and 10× for the application considered in this case study). They are thus preferred resources to reduce both energy supply and cooling costs in the huge datacenters that will be needed by Smart Cities. In this paper, we describe efficient implementations of high-performance algorithms that can process traffic camera image sequences to provide traffic flow information in real-time at a low energy and power cost.

Keywords Smart city · Image processing · Background subtraction · Lucas–Kanade · High-level synthesis · Field programmable gate array · Graphical processing unit

1 Introduction

Cities are seeing massive urbanization worldwide, thus increasing the pressure on infrastructure to sustain private and public transportation. Adding intelligence to traditional traffic management and city planning strategies is essential to preserve and even improve quality of life for citizens under this enormous increase of population. Traffic causes increased delays, thus reducing the opportunity for city dwellers to earn money by performing productive activities. It also poses health hazards due to pollution and

accidents. Several public and private entities (ranging from public transportation providers, to city planners, to traffic light control, to taxi and car sharing providers, to individual drivers) can profit from the widespread availability of real-time information about traffic flows.

The main aim of this paper is to present a computer vision application, which operates in the Smart city context. This application will provide cost-effective and scalable real time analysis of traffic in cities that can then be harnessed by other smart city services and applications (e.g., intelligent traffic management tools) to reduce traffic-related impacts on the quality of life of citizens. Videos obtained from cameras can provide reliable information about the traffic flow on roads. The basic idea, as shown in Fig. 1 is that the cameras acquire the images, which are then processed using image-processing algorithms. After that, the data are stored in a database and accessed on demand.

✉ Arslan Arif
arslan.arif@polito.it

¹ Politecnico Di Torino, Turin, Italy

² ACCIONA Infrastructure S.A., Madrid, Spain

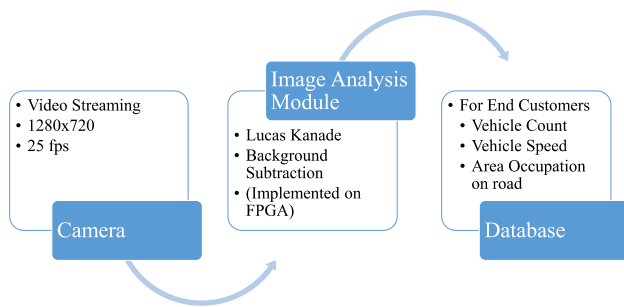


Fig. 1 Application overview

However, the use of cameras poses some disadvantages. The first major drawback is the breach of privacy. Citizens usually feel uncomfortable and insecure when their movements are being monitored and they tend to oppose any such system. To overcome this disadvantage, the end users of our application are not given the raw data. Rather they are provided with only the result of the processing of the images recorded by the cameras. This ensures both the protection of personal information and the value of data.

Another difficulty in the use of such systems is the huge effort required to compute and process data by image analysis algorithms. For instance, cameras should be deployed every 50 m or so to obtain a density that can provide complete information for a city. A big city with an urban area of 360 km² would require the use of about 100,000 active cameras. This can be supported only by extreme parallel computing techniques.

Two commonly used accelerators in the field of parallel computing are Graphical Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). They provide a good solution to achieve high computational power. Both options have their advantages and disadvantages. GPUs are power hungry, whereas developing complex applications for FPGAs using Hardware Description Languages (HDL) is difficult and time consuming. With the introduction of techniques such as High-Level Synthesis (HLS), the effort of programming FPGAs has been significantly reduced and their low energy consumption makes them a great candidate for such large scale applications.

One more point to keep in mind when planning for such systems is that cities tend to grow. Therefore, our system architecture is designed to be scalable, i.e., to allow cameras to be added as needed. Scaling the number of cameras is crucial to make this system practical.

The rest of the paper is organized as follows. Section 2 discusses the previous work in this field. Sections 3 and 4 give an overview of the application and explain the selection of specific image processing algorithms. Section 5 discusses the application constraints, whereas Sect. 6

discusses the Hardware computation performance and costs. The work is concluded in Sect. 7.

2 Related work

A lot of work has been carried out on smart cities in the last 20 years [1]. For some reviewers, smart cities are still confusing [2]. Definitions range from information and communication technology (ICT) networks in city environments [3] to various ICT attributes in a city [4]. Some relate the term with indexes such as the level of education of citizens or in terms of financial security [5], while others think about it in terms of urban living labs [6]. All of these implications are alternative schools of thought and most researchers point towards the complexity and scale of the smart city domain [7].

The monitoring of roads for security and traffic management purposes is one of the main topics in this domain. Modern smart cities measure the traffic so that they can optimize the utilization of the roads and streets by taking actions which can improve traffic flow. Video-based approaches have been researched to monitor the flow of vehicles to obtain rich information about vehicles on roads (speed, type of vehicle, plate number, color, etc.) [8].

Vision-based traffic monitoring applications have seen many advances thanks to several research projects that were aimed at improving them. In 1986, the European automotive industry launched the PROMETHEUS European Research Program [9]. It was a pioneer project which intended to improve traffic efficiency and reduce road fatalities [10]. Later, the Defense Advanced Research Projects Agency introduced the VSAM project to create an automated video understanding technology which can be used in urban and battlefield surveillance applications of the future [11]. Within this structural framework, a number of advanced surveillance techniques were demonstrated in an end-to-end testbed system which included tracking from moving and stationary camera platforms and real-time moving object detection as well as multi-camera and active camera control tracking techniques. The cooperative effort of these two pioneering projects remained active for about two decades. As a result, new European frameworks evolved to cover a variety of visual monitoring systems for road safety and intelligent transportation. In the early 2000s, the ADVISOR project was implemented successfully to spot abnormal user behaviors and develop a monitoring system for public transportation [12–14].

There are several methods which can extract and classify raw images of vehicles. These methods are chiefly feature-based and require hand-coding for detection and classification of specific features of each kind of vehicle. Tian et al. [15] and Buch et al. [8] surveyed some of these methods. In

the fields of intelligent transportation systems and computer vision, intelligent visual surveillance plays a key role [16]. An important early task is foreground detection, which is also known as background subtraction. Many applications such as object recognition, tracking, and anomaly detection can be implemented based on foreground detection [17, 18].

An application was proposed in the Artemis Arrowhead Project [19] that can detect patterns of pedestrians and vehicles. According to the authors, based on this information, the application can also extract a set of parameters such as the density of vehicles and people, the average time during which the elements remain stationary, the trajectories followed by the objects, etc. Subsequently, these parameters are offered as a service to external parties, such as public administrations or private companies that are interested in using the data to optimize the efficiency of existing systems (e.g., traffic control systems or streetlight management) or develop other potential applications that can take advantage of them (e.g., tourism or security).

Many existing systems, which are concerned about privacy of the citizens, employ some sort of censorship so that human or AI users are not able to see and inadvertently recognize any person in the camera footage. This can be done either in the form of a superimposed black box, which blocks out the eyes or face of the person, masking each person in each frame or blocking images of certain places altogether [20–25]. However, this approach cannot achieve full privacy. Most of the time we do not require any sort of information related to individuals while working with applications related to computer vision. Thus, the developer should be aware of the information being collected either advertently or inadvertently and of what are the real requirements for the application [26].

Extraction and categorization of vast amounts of data require expensive and sophisticated software. Processing the live feed for even a single camera requires a dedicated CPU [27]. More performance requires computer accelerators. The most commonly used computer accelerator in this domain is the Graphical Processing Unit (GPU). GPUs provide higher memory bandwidth, higher floating point throughput and a more favorable architecture for data parallelism than processors. Due to these properties, they are used in modern high-performance computing (HPC) systems as accelerators [28]. However, the main drawback of HPC systems based on GPU accelerators is that they consume large amount of power [29].

To overcome the power inefficiency of GPU-based HPC systems, modern field programmable gate arrays (FPGAs) can be used. FPGA devices require less operating power and energy per operation while providing reasonable processing speed as compared to GPUs [30]. When comparing them with multi-core CPUs, especially with regards to data center applications, it was observed that the performance gap keeps

widening between the two. In summary, FPGAs are known to be more energy efficient than both CPUs and GPUs [31]. Acknowledging these capabilities, Microsoft, Baidu and Amazon now also use FPGAs as accelerators rather than GPUs in their data centers [32].

FPGAs are, however, complex to program. Hardware description languages (HDL) such as Verilog or VHDL are commonly used for this task. A technique called high-level synthesis (HLS) provides the capability to program FPGAs through the use of high-level languages, e.g., C, C++, OpenCL or SystemC, consequently reducing the design time debugging and analysis [33, 34].

3 The application

The main goal of the application described in this paper is to extract data from video surveillance cameras and make it available to different services. The objective is to provide real-time information which can be used to optimize, for example, the street lighting and traffic light systems installed in cities. The application will analyze the images recorded by the cameras installed in cities and will apply a set of algorithms to detect the presence of people and vehicles and to compute the density of traffic at each specific location.

For this purpose, cameras are installed on roads (Fig. 2). Their parameters such as height from ground, angle of elevation, and road parameters such as width, are already assumed to be available for processing, as shown in Fig. 3, together with other constants such as the minimum value for detecting a change of speed.

In most places, cameras cannot be positioned directly above a road. Most of the times they will have a prospective view, as shown in Fig. 2. So we need input values to map the road with respect to the camera pixels (Fig. 4). We need three types of information. (1) Whether a pixel covers a road area, (2) how much area each pixel covers and (3) how much distance each pixel covers in the direction of the camera. The presence or absence of the road



Fig. 2 Camera view

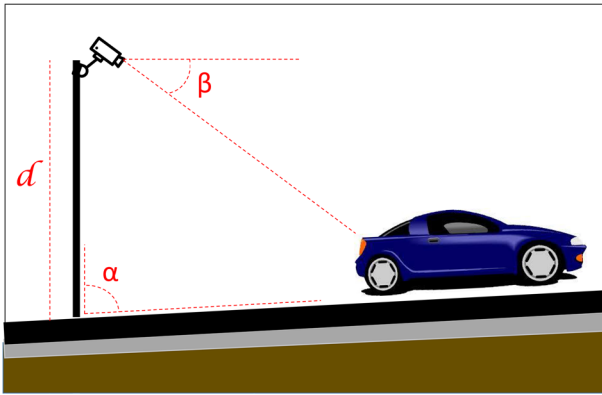


Fig. 3 Road parameters w.r.t camera

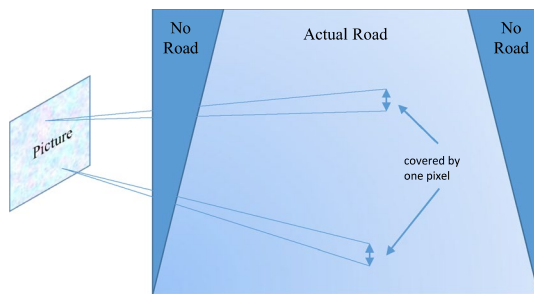


Fig. 4 Video frame vs ground reality

allows us to apply the algorithm only on the part of the camera frame that we are interested in and hence save computational resources. The area value is used to find the percentage of the road occupied by moving objects. Finally the distance is used to compute the velocity of the vehicles. All of them can be calculated from camera resolution, aperture, focal length and height over the road. Another important thing to note here is that, as we move away from the camera, the distance represented by one pixel increases. Therefore, the distance value for each pixel is different. It is calculated once for each stationary camera and then used repeatedly to save time and computational resources.

Figure 5 shows the general workflow of the image analysis module in detail. Two configuration files containing road and camera parameters are used as inputs, in addition to the image to be analyzed. This module can be instantiated, as many times as needed, once for each descriptor that is desired, so that it is possible to detect many kinds of objects at the same time.

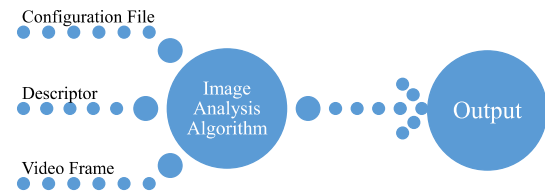


Fig. 5 General workflow of image analysis module

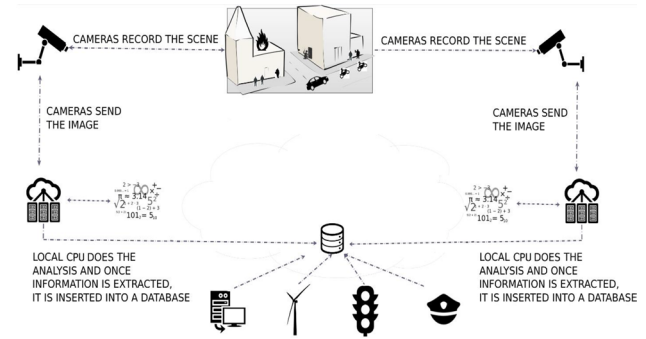


Fig. 6 Decentralized model

3.1 Implementation model

Two types of implementation are possible for this system on the basis of the location of computational and storage units. One is decentralized, where each camera has its own processing unit. The other is centralized, where all the processing by a set of closely situated cameras is done on one single server.

3.1.1 Decentralized architecture

Figure 6 represents the decentralized architecture version of the application. Due to the high computational requirements, a dedicated CPU would be needed for each camera installed in the monitored scenario. Once the image (which must be processed in real time) is captured, the pre-processing unit associated to that camera processes the signal for detecting the elements present in the image. Afterwards, it sends a picture with some metadata to the central processing unit in which all of the information are processed and stored to be offered to the customers within a cloud architecture.

3.1.2 Centralized architecture

On the other hand, Fig. 7 depicts an architecture in which one processing unit is used by a number of cameras. The idea is to combine the processing unit with the central database where all the data are offered to the customer. This

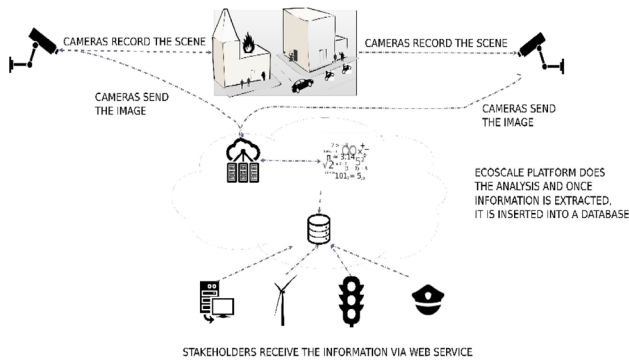


Fig. 7 Centralized model

means that no camera has a dedicated processing unit attached, which dramatically increases the amount of data to be processed centrally in real time.

After analyzing both options, the second alternative is considered more appropriate because of the costs of implementation, application software management, maintenance costs to resolve hardware failures, improved safety, etc. In Fig. 8, the scheme for the proposed solution is presented. A major factor for choosing a centralized system would be the achievable energy efficiency using latest generation FPGA devices, which are very power-efficient but too expensive to be deployed in a decentralized architecture.

Most of the operations carried out in image processing are pixel based, with no or very few dependencies on other pixel output values. This provides a very good basis for a parallel implementation of image-processing algorithms that work on each pixel either simultaneously or in a pipelined fashion (Fig. 9). In this way, we can reduce the frame processing time and hence we can achieve a real time processing frequency, which is about 25 fps for the target application.

3.2 Proposed architecture

We target to provide an energy-efficient architecture by sharing numerous reconfigurable accelerators. To provide a scalable approach, the architecture should be tailored to the

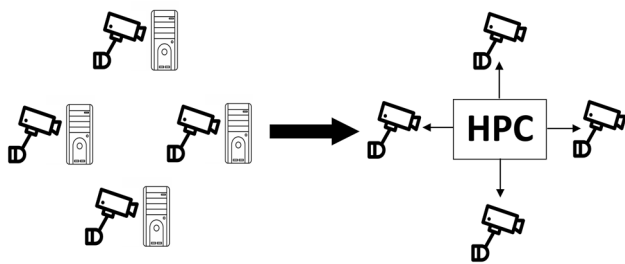


Fig. 8 From decentralized to centralized architecture

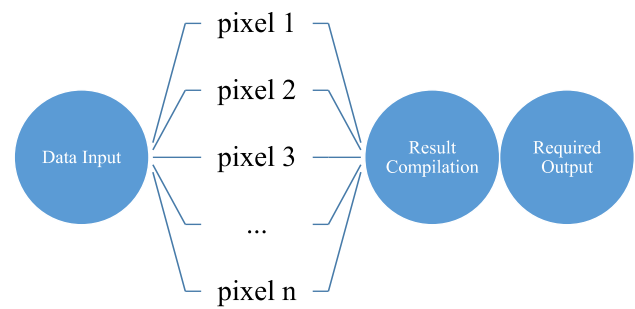


Fig. 9 Overview of parallelism in image-processing algorithms

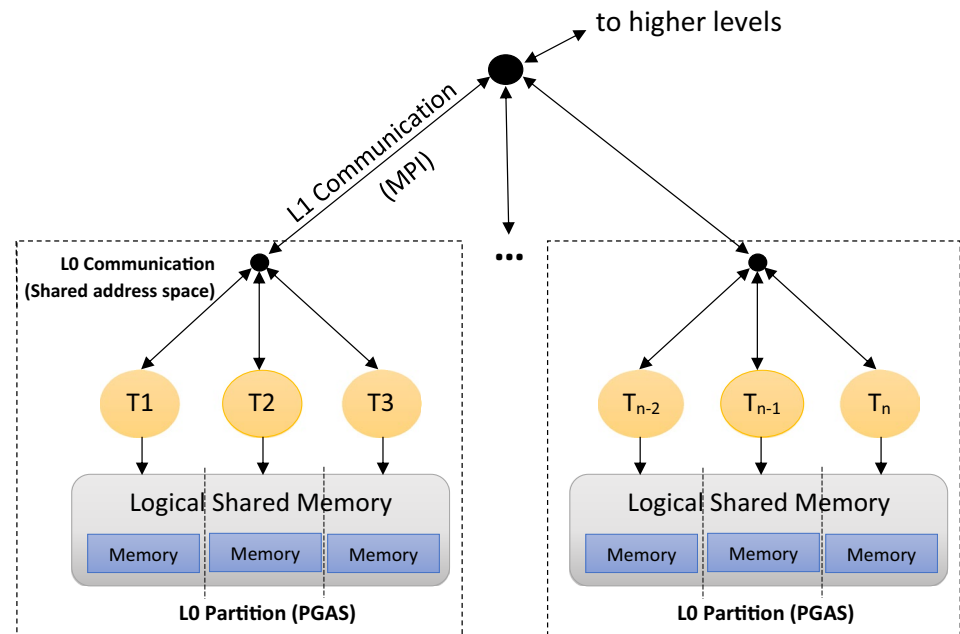
needs of the HPC applications as well to the characteristics of the hardware platform. Energy-efficient heterogeneous Computing at exaSCALE (ECOSCALE) is a project under the H2020 European research framework. The main goal of this project is to provide a hybrid MPI + OpenCL programming environment, a hierarchical architecture, a runtime system and middleware, and a shared distributed reconfigurable FPGA-based acceleration [35].

ECOSCALE offers a hierarchical heterogeneous architecture with the purpose of achieving exascale performance in an energy-efficient manner. It proposes to adopt two key architectural features to achieve this goal: UNIMEM and UNILOGIC. UNIMEM was first proposed by the EURO-SERVER project [36] and provides efficient uniform access, including low-overhead ultra-scalable cache coherency, within each partition of a shared Partitioned Global Address Space (PGAS). UNILOGIC, which is first being proposed by ECOSCALE, extends UNIMEM to offer shared partitioned reconfigurable resources on FPGAs. The proposed HPC design flow, supported by implementation tools and a runtime software layer, partitions the HPC application design into several nodes. These nodes communicate through a hierarchical communication infrastructure as shown in Fig. 10. Each Worker node (basically, an HPC board) includes processing units, programmable logic, and memory. Within a PGAS domain (several Worker nodes), this architecture offers shared partitioned reconfigurable resources and a shared partitioned global address space which can be accessed through regular load and store instructions by both the processors and the programmable logic. A key goal of this architecture is to be transparently programmable with a high-level language such as OpenCL.

4 Implemented algorithms

As discussed before, computational accelerators must be used to extract the required information from videos with sufficient performance and energy efficiency. The computing power of the hardware accelerators will focus on the vision

Fig. 10 Hierarchical partitioning (tasks, memory, communication) of an HPC application [35]



algorithms for recognition and measurement of traffic, as they are the most expensive part of the application. Two approaches which are best suited for our application have been identified for processing the images streamed from fixed cameras.

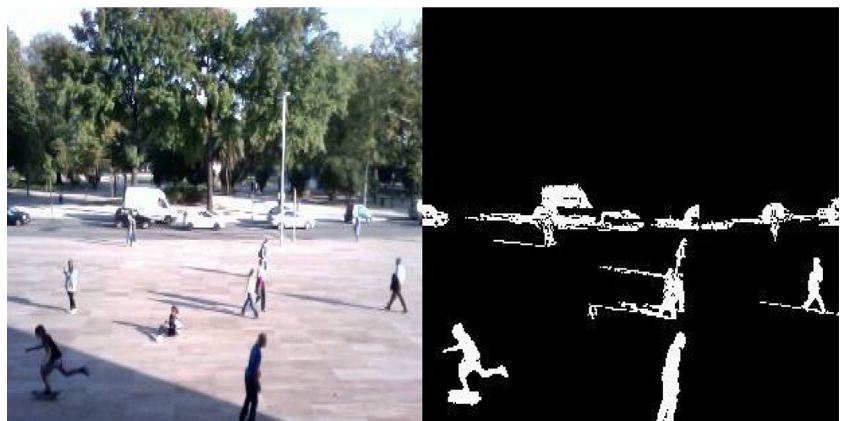
The algorithms are coded in the OpenCL language. OpenCL is a programming language for parallel architectures which is built upon C/C++ and thus can be easily learned and ported [37]. The basic advantage of OpenCL is that it can exploit the architectural features of accelerators more easily than C or C++. It provides the programmer with a clear distinction between different kinds of memory, such as global DRAM, local on-chip SRAM and private register files. This allows programmers to optimize code much better than with the flat memory models of Java C and C++.

4.1 Vehicular density on the roads

Algorithm 1 is based on a background subtraction and object tracking method. One popular implementation was made available by Laurence Bender et al. as part of the SCENE package [38], available in the SourceForge repository (Fig. 11). The algorithm performs motion detection principle by calculating the change in the corresponding pixel values with respect to the reference stationary background. The portion of the road where movement is detected gives an idea about the amount of traffic. Moreover, the algorithm also constantly updates the reference background image (in case a moving object is now at rest).

Our chosen algorithm takes four frames (images) as input, including the reference stationary background, the frame under the consideration, the preceding frame and the

Fig. 11 Output of the background subtraction algorithm [38]



succeeding frame. For each pixel, it performs a weighted difference on the corresponding pixels of three consecutive frames. If this difference is zero, it implies that there is no movement in the corresponding pixel, hence no update is needed for the total moving area or the reference background. On the other hand, non-zero values corresponds to some change in the consecutive video frames around the pixel. The value can be a positive or a negative number according to the direction of movement with respect to the camera. If the absolute of this value is larger than the threshold set for movement detection and some change is also detected in the current frame pixel w.r.t. the reference background, then the global accumulator of the moving area is updated by adding the area of the road occupied by the current pixel. If the weighted difference is less than the threshold for $N - 1$ frames, then the algorithm updates the reference background pixel with the current pixel. N is the minimum number of frames required to declare the pixel to be part of the stationary background. The value of N can be set according to the application.

Algorithm 1 Background Subtraction algorithm

Require: Four grayscale images $image_{-1}$, $image_0$, $image_1$ and $image_{bg}$ & Count array

Ensure: $image_{out}$, Updated $image_{bg}$ and Count array & Total Area with Movement

```

1: for  $j = 0$  to  $HEIGHT - 1$  do
2:   for  $i = 0$  to  $WIDTH - 1$  do
3:      $PIX = (j * WIDTH) + i$ 
4:      $lat = 0$ 
5:     if  $PIX$  is on ROAD then
6:        $center \leftarrow PIX$ 
7:        $left \leftarrow PIX - 10$ 
8:        $right \leftarrow PIX + 10$ 
9:        $lat \leftarrow Abs(\text{sum of weighted difference of } left, right \text{ and } center \text{ pixels of all three images})$ 
10:    end if
11:    if  $(lat < \text{threshold}) \ \& \ (\text{Count}[PIX] \geq N)$  then
12:       $image_{bg}[center] \leftarrow image_0[center]$ 
13:    else
14:       $\text{Count}[PIX]++$ 
15:    end if
16:    if  $((image_0[center] - image_{bg}[center]) > \text{Background threshold}) \ \& \ (lat > \text{threshold})$  then
17:       $image_{out}[center] \leftarrow image_0[center]$ 
18:      Increment Area with Movement
19:    else
20:       $image_{out}[center] \leftarrow 0$ 
21:    end if
22:  end for
23: end for

```

4.2 Vehicular velocity on the roads

Since the background subtraction module can only find the area occupied by moving objects on the roads, another method is needed to measure the velocity of vehicles, based

on the Lucas–Kanade algorithm for optical flow [39]. An implementation of the Lucas–Kanares optical flow algorithm developed by Altera [40] in OpenCL with a 52×52 window size is shown in Fig. 12.

A window size of $N \times N$ means that the optical flow for one pixels is computed with respect to the neighboring $N/2$ pixels on each side of that pixel, i.e., the pixel under consideration is in the center of a matrix of pixels having $(N+1)$ rows and columns. For each pixel in the window, a partial derivative with respect to its horizontal (I_x) and vertical (I_y) neighbors is computed. The size of the window is a compromise between true negative and false positive change detection. Therefore, it should be chosen by an expert with respect to area covered by each pixel and other parameters. In this paper, we use a 15×15 window.

A pyramidal implementation [41] is used to refine the optical flow calculation and the iterative Lucas–Kanade optical flow computation is used for the core calculations. For each pixel, computed partial derivatives within the window and the difference among the pixel values in the current and next frames are used to calculate the velocity of each moving object (it is zero if the area covered by the pixel is stationary). The magnitude is the speed of the object, whereas the sign shows whether it moves towards the camera or away from it.

In our implementation of the algorithm (Algorithm 2), the optical flow is computed for all the pixels of the image (in this case for a 1280×720 resolution). Two images using 8 bits per pixel are compared with a window size of 15. Moreover, the obtained values are mapped to a single color representing both relative velocity and direction, as shown in Fig. 13.

To calculate the average velocity of traffic with the optical flow algorithm one needs to know the distance between the camera and the recorded objects. To avoid expensive and complex solutions for a real-time depth measurement, an approximation for calculating the distance corresponding to each pixel of the image is used based on static camera parameters, such as road plane inclination, camera orientation and field of view.

In addition to the capabilities summarized above, additional features for user interaction are included in the application. For example, a module for defining the target areas where the recognition is performed and setting up the parameters of the different cameras has been developed. All these parameters can be given as an input in the configuration file.

Fig. 12 Altera's implementation of Lucas–Kanade algorithm [40]



Algorithm 2 Lucas–Kanade algorithm

Require: two frames of images $image_0$ and $image_1$ and other coefficients

Ensure: v_{opt}

```

1: for  $j = 0$  to  $HEIGHT - 1$  do
2:   for  $i = 0$  to  $WIDTH - 1$  do
3:      $G_{2 \times 2} \leftarrow 0$ 
4:      $b_{2 \times 1} \leftarrow 0$ 
5:     for  $w_j = -w_y$  to  $w_y$  do
6:       for  $w_i = -w_x$  to  $w_x$  do
7:          $center \leftarrow Pos(i + w_i, j + w_j)$ 
8:          $left \leftarrow Pos(i + w_i - 1, j + w_j)$ 
9:          $right \leftarrow Pos(i + w_i + 1, j + w_j)$ 
10:         $up \leftarrow Pos(i + w_i, j + w_j - 1)$ 
11:         $down \leftarrow Pos(i + w_i, j + w_j + 1)$ 
12:         $im_{val}^0 \leftarrow image_0[center]$ 
13:         $im_{val}^1 \leftarrow image_1[center]$ 
14:         $\delta I \leftarrow d(im_{val}^0, im_{val}^1)$ 
15:         $im_{left}^0 \leftarrow image_0[left]$ 
16:         $im_{right}^0 \leftarrow image_0[right]$ 
17:         $I_x \leftarrow (im_{right}^0 - im_{left}^0)/2$ 
18:         $im_{up}^0 \leftarrow image_0[up]$ 
19:         $im_{down}^0 \leftarrow image_0[down]$ 
20:         $I_y \leftarrow (im_{down}^0 - im_{up}^0)/2$ 
21:         $G \leftarrow G + g_{2 \times 2}(I_x, I_y)$ 
22:         $b \leftarrow b + f_{2 \times 1}(\delta I, I_x, I_y)$ 
23:      end for
24:    end for
25:     $G \leftarrow inverse(G)$ 
26:     $v_{opt}[j][i] \leftarrow G \times b$ 
27:  end for
28: end for

```

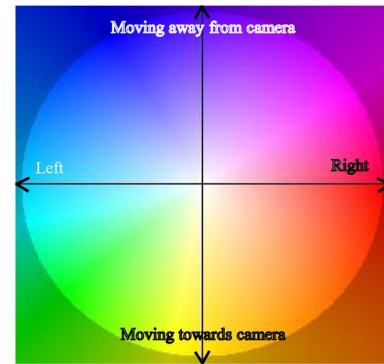


Fig. 13 Lucas–Kanade's disparity map

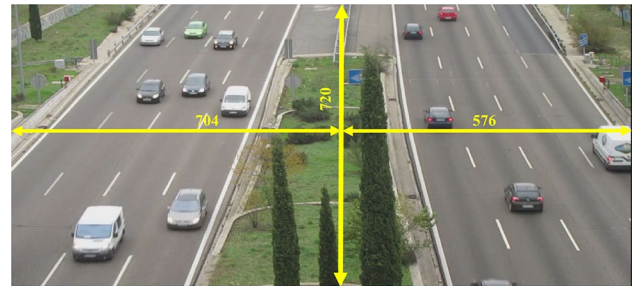


Fig. 14 Sample frame

5 Application constraints

As discussed before, we are dealing with live video streaming in our application. The cameras that we are using produce 25 frames per second (fps) with an image resolution of 1280×720 pixels. These frames are given as input to both image-processing algorithms explained in section IV, one for moving object detection and one for speed estimation. A sample frame from one of the cameras is shown in Fig. 14.

5.1 Background subtraction algorithm

The background subtraction algorithm needs three consecutive frames and a reference stationary background image to distinguish between moving and stationary objects. After the computation of one set of frames, the next frame is fed to the kernel and the oldest one is removed from the set. The result is shown in Fig. 15.

Here the static areas are detected as background and converted to black, while pixels where movements have been detected are shown as gray-scale pixels of the original frame. We also compute the portion of the road that is occupied by



Fig. 15 Output of background subtraction

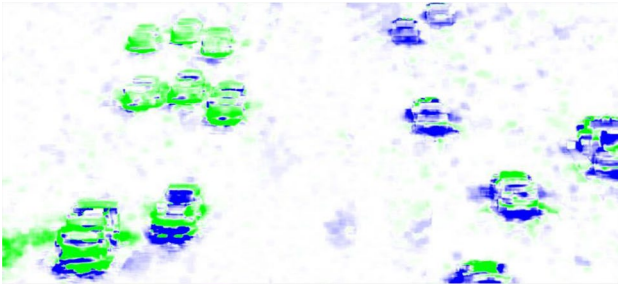


Fig. 16 Output of Lucas–Kanade algorithm

moving objects. In this set of frames, it is equal to 11.2 m^2 on the side where traffic is coming towards the camera, and it is 6.55 m^2 on the side where traffic is moving away from the camera.

5.2 Lucas–Kanade algorithm

In our implementation of the Lucas–Kanade Algorithm, for each set of calculations, we need two consecutive image frames and a set of input parameters depending on the road conditions and camera angles. Similar to background subtraction, each new frame replaces the older one. The graphical output from these images is shown in Fig. 16. The stationary regions are represented by white pixels, while moving objects are mapped to colors according to their speed and direction.

A interesting result is the speed of moving objects (vehicles) on the road. For the current frame as reference. The average velocity coming towards the camera is about 118 km/h while the velocity moving away is -67 km/h . The direction of the vehicles is evident also from the color in Fig. 16, in accordance with the encoding shown in Fig. 13.

We can also find the speed in any specific lane of the road, by dividing the pictures in separate lanes instead of two parts as we did in Fig. 14. This can be achieved, if required, by minor adjustments in the input configuration file.

Note that the processed images or data extracted from them contain no personal information, thus we can safely say that we have achieved the objective of personal data integrity and we are not forwarding any sort of personal or privileged information to any third party.

6 Implementation results and algorithm optimization

After testing the basic functionality of the algorithms, we optimized them to get the maximum efficiency with a minimum use of resources in the smallest amount of computational time. Performance analysis was carried out using RTL simulation on a virtual board including a Virtex 7 FPGA from Xilinx and then on real hardware, using the Amazon Web Services(AWS) Elastic Compute Cloud (Amazon EC2). The available resources on these boards are shown in Table 1. Note that to complete RTL simulations (for Virtex 7) in a reasonable amount of time, we used an image resolution of 1280×4 and we extrapolated the simulation results to the real image size. On AWS, on the other hand, the complete frame was used to verify the results. For high level synthesis, we used SDAccel v2016.4 and 2017.1 from Xilinx.

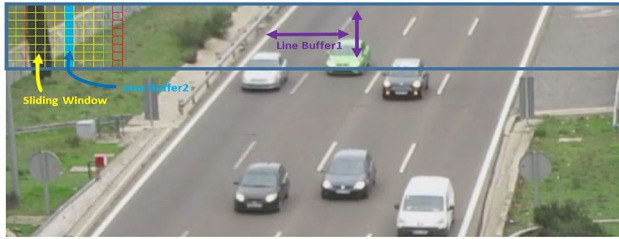
Moreover, simulations were carried out for a single compute unit and then a suitable number of compute units that could fit on the FPGA were used for each algorithm. In contrast to a CPU or GPU, an FPGA does not have a fixed architecture but the HLS tool generates a custom computation and memory architecture from each application. The

Table 1 Target FPGAs and boards

Target device name	ADM-PCIE-7V3:1ddr:3.0	AWS-F1:4ddr-xpr-2pr:4.0
FPGA part (Xilinx)	Virtex-7 XC7VX690T-2	Virtex UltraScale+ xcvu9p-2-i
Clock frequency	200 MHz	250 MHz
Memory bandwidth	9.6 GB/s	11.25 GB/s
BRAMs	2940	4320
URAMs	—	960
DSPs	3600	6840
FFs	866,400	2,364,480
LUTs	433,200	1,182,240

Table 2 Kernel execution time and resource utilization (per compute unit) of background subtraction algorithm

Implementation version	Time (ms)	Resource utilization			
		Per frame	BRAM	DSP	FF
Basic	7313.112	5	3	14,447	35,019
Optimized v.1	1467.108	22	5	10,979	31,700
Optimized v.2	103.8096	24	5	9165	15,978
Virtex 7 (3 CU)	34.6032	72	15	27,495	47,934
UltraScale+ (3 CU)	27.8082	65	15	18,723	17,859

**Fig. 17** Line buffers for Lucas–Kanade

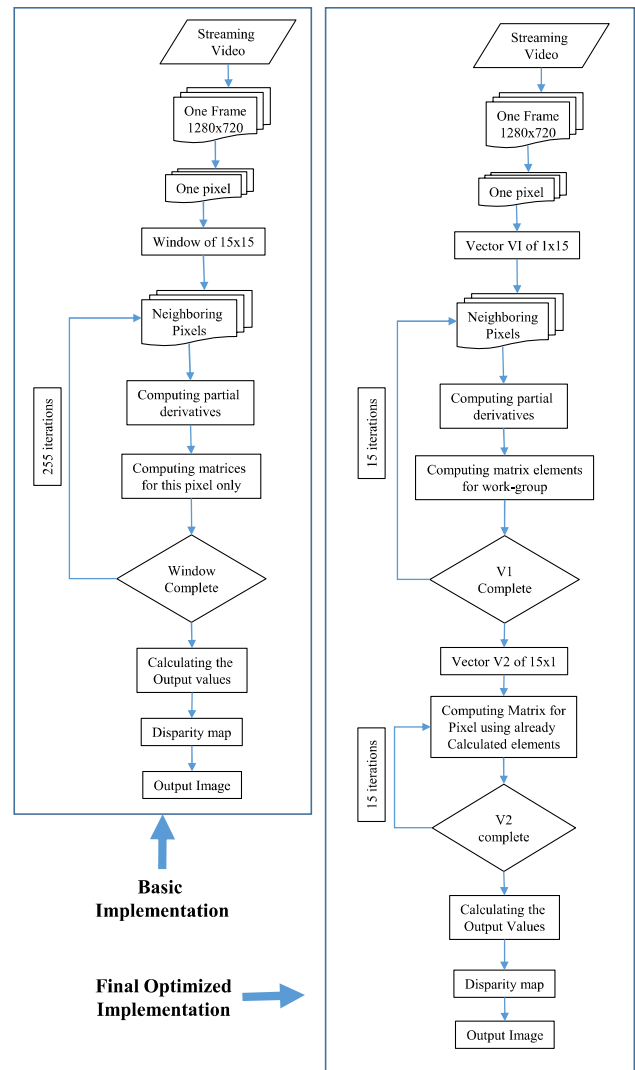
term “compute unit” (CU) refers to a specialized hardware architecture (processing core) for a given application. The designer can use multiple parallel CUs (within the available resources) to boost the performance of each application.

The application needs to process 25 frames per second to meet the requirement of real time video processing. This means that each kernel iteration (processing one frame) should be completed in a maximum time of 40 m.

6.1 Background subtraction algorithm

The initial implementation of the background subtraction algorithm was faster than the optical flow algorithm, but still did not match the real time requirements. The bottleneck for this algorithm was global memory access. To solve this issue, a line buffer was introduced. The kernel fetches all the pixel values required for each work group and stores them in a line buffer in local memory. This fetching is implemented using the OpenCL asynchronous work group copy operation, which is implemented as a burst read operation from DRAM to on-chip memory (much faster than single transfers). The same mechanism is used for burst writes. This reduces the kernel execution time by a factor of 5 but increases BRAM utilization. The results are good but still the desired pipelining of work items is not achieved due to the read/modify/writes required to update global variables such as the total moving area.

In the second version, local buffers are also used for the standard background image, the array accounting for the number of frames with a slight change and the global

**Fig. 18** Basic vs final implementation of Lucas Kanade**Table 3** Kernel execution time and resource utilization (per compute unit) of Lucas–Kanade algorithm

Implementation version	Time (ms)	Resource utilization			
		Per frame	BRAM	DSP	FF
Basic	44,209.98	31	56	21,367	37,080
Optimized v. 1	14,883.876	122	51	18,410	24,613
Optimized v. 2	3751.2	182	52	51,025	100,777
Optimized v. 3	207.313	178	175	35,683	36,072
Virtex 7 (6 CU)	34.5522	1068	1050	214,098	216432
UltraScale+ (6 CU)	39.4512	1386	576	274,770	246,786

accumulator of the moving area which were causing the bottleneck in the first place. In this way, we are able to achieve the expected performance, a speed gain of more than 70x

Table 4 Total resource utilization for Virtex 7

Algorithm	Compute units (CU)	Total resources utilized			
		BRAM	DSP	FF	LUT
Background subtraction	3	72	15	27,495	47,934
Lucas–Kanade	6	1068	1050	214,098	216,432
Total	9	1140	1065	241,593	264,366
Available	–	2940	3600	866,400	433,200

Table 5 Total resource utilization for UltraScale+ (AWS-EC2)

Algorithm	Compute units (CU)	Total resources utilized			
		BRAM	DSP	FF	LUT
Background subtraction	3	65	15	18,723	17,859
Lucas–Kanade	6	812	246	176,970	168,280
Total	13	877	261	195,693	186,139
Available	–	4320	6840	2,364,480	1,182,240
% Utilization	–	20.30%	3.81%	8.27%	15.74%

from the basic implementation and more than 14x from our first optimized version. The extra resources consumed are only two BRAMs.

However, the best time that we achieved using Hardware emulation was 103 ms per frame, hence not sufficient to achieve 25 fps. For this purpose, we need to use at least three parallel compute units, which multiplies all the resources by a factor of 3 as shown in Table 2. This still uses only about 12% of the resources of a Virtex 7 FPGA, which can thus processes frames from five cameras. The results obtained from AWS EC2 board show an increase in performance which was expected as UltraScale+ is a newer generation FPGA than Virtex 7. These results are shown in the last row of Table 2.

6.2 Lucas–Kanade algorithm

The basic implementation of the Lucas–Kanade Algorithm is even more costly than the background subtraction algorithm. Three main opportunities for optimizing were global memory access, avoiding repeated calculations for the same pixel and optimizing trigonometric calculations for the output colors.

The first optimized version of the kernel uses a line buffer for burst reading and writing of the image data from global to local memory (similar to what we have seen in the background subtraction algorithm). For Lucas–Kanade this line buffer is about five times larger than what we used in background subtraction because more neighboring pixels are required for computation. This can be easily seen by the increase in the number of BRAMs (about four times) in the first version as compared to the basic implementation.

Since the partial derivative calculated for a pixel in the window is also required by the next 14 windows (using a

Table 6 Power consumption per frame for background subtraction

Parameters	FPGA		GPU	CPU
	Ultrascale+	Virtex 7		
Device time (ms)	27.80	34.6	28.16	47.68
Device power (W)	4.55	2.760	26	10
Energy (mJ)	126.49	95.496	732.16	476.8

sliding window as shown in Fig. 17), in the second optimized version we removed this repetitive computation by calculating it only once and reusing it (Line Buffer 2 in Fig. 17). In this way, we not only saved computations per work group, but also were able to split the loop nest (line 4 and 5 of Algorithm 2) into two single loops as shown in Fig 18. This reduces the iterations from 225 (15×15) to 30 ($15 + 15$). A work-group size of 1280 was also used, as it avoids not only the repetitive fetching of neighbors among work groups (along the width of image) but also eliminates repetitive calculations for each WG. This gives us a performance boost of 4x but also requires a lot more resources (Table 3).

The analysis of the second optimized version revealed that the algorithm is not able to pipeline the inner loop because of the trigonometric functions for the output color encoding. These calculations were required only for debugging. Since the information provided to end users is purely average velocity on each lane, therefore the Lucas–Kanade algorithm debug image is calculated in a simpler way. For debugging, the most interesting part of the image is the one that closest to the line of sight. Hence, it is possible to use a linear pixel mapping, rather than using trigonometric functions, which are expensive to compute just for debugging and system monitoring

Table 7 Power consumption per frame for Lucas–Kanade algorithm

Parameters	FPGA		GPU	CPU
	Ultrascale+	Virtex 7		
Device time (ms)	37.31	36.34	42.68	5925.78
Device power (W)	8.0	8.385	75	10
Energy (mJ)	298.48	304.7	3201	59,257.8

purposes on an FPGA. This resulted in a degraded depiction of sideways motion, but overall improved the FPGA execution time by 15× (Table 3). Hence, it shows that floating point computations are FPGA's weakest point. To satisfy real-time requirements, we have to use six Compute Units for the core calculations of the Lucas–Kanade algorithm.

As we witnessed from background subtraction as well, the results obtained from AWS EC2 for the Lucas–Kanade algorithm are very comparable to the hardware emulation results as shown in Fig. 18. In both cases performance improved and the amount of available resources increase significantly on a Virtex Ultrascale+ with respect to the Virtex 7. Hence, we were able to feed the data from four cameras in real-time to the EC2 board.

6.3 Total resource utilization and power consumption

Summing up all the results discussed above, we achieved our goal of real time calculation of the portion of the road that is used by traffic and of average vehicular velocity. Moreover, Table 4 shows that we have not exceeded our resource utilization limit, while performing the full processing of the data from one camera on a relatively old Virtex 7 FPGA. The results of actual Hardware implementation on the Amazon EC2 cloud platform are shown in Table 5.

The final aspect to consider is what advantage we have achieved in terms of power and energy consumption (per computation) with respect to GPUs and CPUs. We are considering an NVIDIA GeForce GTX960 GPU. It has 2GB of global memory and bandwidth of 112 GB/s with a maximum power consumption of 120 Watt. The CPU that we are considering is an Intel Xeon E3-1241 (v3) with a clock frequency of 3.5 GHz and maximum power consumption of 80 Watts. The power consumption for the FPGAs was estimated using the Xilinx Power Estimator (XPE) tool while for the GPU it was measured using NVIDIA System Management Interface (NVIDIA-SMI).

As we can see from Tables 6 and 7, the FPGA is much more energy efficient as compared to both CPU and GPU. Moreover, the computation of Lucas Kanade is not possible in real time using only a single CPU, as it takes around 6 s to process each frame. As we can see both, performance

and energy consumption, are much better than on a CPU and energy consumption is much better than on a GPU.

7 Conclusion

This paper presents a high performance yet energy efficient smart city application implementation. The application provides not only the velocity of the vehicles in real time but also the density of traffic on roads. This information can be used by different stake holders such as public transportation, taxis and city planners. Real-time benefits of these data can save time spent on roads and can help to reduce pollution where in long run these data can be used for better planning of city and road infrastructure. The computational capabilities and power efficiency of FPGAs makes them a very suitable candidate for applications that require large amounts of data processing, especially in real time. Furthermore, high-level synthesis provides an excellent platform for designers to exploit the capabilities of FPGAs without the long design times entailed by the use of in hardware description languages.

Acknowledgements This work is supported by the European Commission through the H2020 ECOSCALE project (Project ID 671632).

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

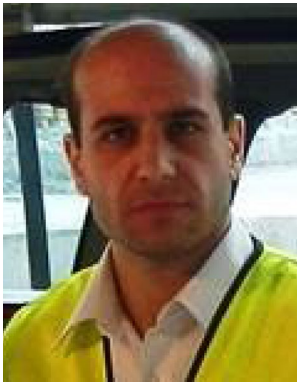
References

1. Akçura, M.T., Avci, S.B.: How to make global cities: information communication technologies and macro-level variables. *Technol. Forecast. Soc. Chang.* **89**, 68–79 (2014)
2. Anderson, J., et al.: Getting smart about smart cities: understanding the market opportunity in the cities of tomorrow (2012). <http://www.alcatellucent.com>
3. Allwinkle, S., Cruickshank, P.: Creating smart-er cities: an overview. *J. Urban Technol.* **18**(2), 1–16 (2011)
4. Anthopoulos, L., Fitsilis, P.: Using classification and roadmapping techniques for smart city viability's realization. *Electr. J. e-Gov.* **11**(2), 326–336 (2013)
5. Anthopoulos, L.G., Tsoukalas, I.A.: The implementation model of a digital city. the case study of the digital city of Trikala, Greece: e-Trikala. *J e-Gov.* **2**(2), 91–109 (2006)
6. Komninos, N.: *Intelligent cities: innovation, knowledge systems, and digital spaces*. Taylor and Francis, Boca Raton (2002)
7. Anthopoulos, L.G.: Understanding the smart city domain: a literature review. In: *Transforming city governments for successful smart cities*. Springer, Berlin, pp 9–21 (2015)
8. Buch, N., Velastin, S.A., Orwell, J.: A review of computer vision techniques for the analysis of urban traffic. *IEEE Trans. Intell. Transp. Syst.* **12**(3), 920–939 (2011)
9. Williams, M.: The prometheus programme. In: *Towards safer road transport-engineering solutions*, IEE colloquium on. IET, London, pp 4-1 (1992)

10. Ulmer, B.: Vita-an autonomous road vehicle (ARV) for collision avoidance in traffic. In: Intelligent vehicles' 92 symposium. Proceedings of the IEEE, Detroit, MI, pp 36–41 (1992)
11. Collins, R.T., Lipton, A.J., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., Tolliver, D., Enomoto, N., Hasegawa, O., Burt, P., et al.: A system for video surveillance and monitoring. VSAM Final Report, pp 1–68. Carnegie Mellon University (2000)
12. Morris, B.T., Trivedi, M.M.: A survey of vision-based trajectory learning and analysis for surveillance. *IEEE Trans. Circuits Syst. Video Technol.* **18**(8), 1114–1127 (2008)
13. Morris, B.T., Trivedi, M.M.: Understanding vehicular traffic behavior from video: a survey of unsupervised approaches. *J. Electron. Imaging* **22**(4), 041 113–041 113 (2013)
14. Datondji, S.R.E., Dupuis, Y., Subirats, P., Vasseur, P.: A survey of vision-based traffic monitoring of road intersections. *IEEE Trans. Intell. Transp. Syst.* **17**(10), 2681–2698 (2016)
15. Tian, B., Morris, B.T., Tang, M., Liu, Y., Yao, Y., Gou, C., Shen, D., Tang, S.: Hierarchical and networked vehicle surveillance in ITS: a survey. *IEEE Trans. Intell. Transp. Syst.* **16**(2), 557–580 (2015)
16. Zhang, J., Wang, F.-Y., Wang, K., Lin, W.-H., Xu, X., Chen, C.: Data-driven intelligent transportation systems: a survey. *IEEE Trans. Intell. Transp. Syst.* **12**(4), 1624–1639 (2011)
17. Yilmaz, A., Javed, O., Shah, M.: Object tracking: a survey. *ACM Comput. Surv. CSUR* **38**(4), 13 (2006)
18. Wang, K., Liu, Y., Gou, C., Wang, F.-Y.: A multi-view learning approach to foreground detection for traffic surveillance applications. *IEEE Trans. Veh. Technol.* **65**(6), 4144–4158 (2016)
19. Jokinen, J., Latvala, T., Lastra, J.L.M.: Integrating smart city services using arrowhead framework. In: Industrial Electronics Society, IECON 2016-42nd annual conference of the IEEE. IEEE, Florence, pp 5568–5573 (2016)
20. Blažević, M., Brkić, K., Hrkać, T.: Towards reversible de-identification in video sequences using 3d avatars and steganography. arXiv preprint [arXiv:1510.04861](https://arxiv.org/abs/1510.04861) (2015)
21. Newton, E.M., Sweeney, L., Malin, B.: Preserving privacy by de-identifying face images. *IEEE Trans. Knowl. Data Eng.* **17**(2), 232–243 (2005)
22. Rashwan, H.A., Solanas, A., Puig, D., Martínez-Ballesté, A.: Understanding trust in privacy-aware video surveillance systems. *Int. J. Inf. Secur.* **15**(3), 225–234 (2016)
23. Raval, N., Srivastava, A., Lebeck, K., Cox, L., Machanavajjhala, A.: Markit: Privacy markers for protecting visual secrets. In: Proceedings of the 2014 ACM international joint conference on pervasive and ubiquitous computing: adjunct publication. ACM, Seattle, pp 1289–1295 (2014)
24. Roesner, F., Molnar, D., Moshchuk, A., Kohno, T., Wang, H.J.: World-driven access control for continuous sensing. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. ACM, Scottsdale, pp 1169–1181 (2014)
25. Schiff, J., Meingast, M., Mulligan, D.K., Sastry, S., Goldberg, K.: Respectful cameras: detecting visual markers in real-time to address privacy concerns. In: Protecting privacy in video surveillance. Springer, Berlin, pp 65–89 (2009)
26. Chen, A.T.-Y., Biglari-Abhari, M., Kevin, I., Wang, K.: Trusting the computer in computer vision: a privacy-affirming framework. In: Computer vision and pattern recognition workshops (CVPRW), 2017 IEEE conference. IEEE, pp 1360–1367 (2017)
27. Engel, J.I., Martin, J., Barco, R.: A low-complexity vision-based system for real-time traffic monitoring. *IEEE Trans. Intell. Transp. Syst.* **18**(5), 1279–1288 (2017)
28. Weber, R., Gothandaraman, A., Hinde, R.J., Peterson, G.D.: Comparing hardware accelerators in scientific applications: A case study. *IEEE Trans. Parallel Distrib. Syst.* **22**(1), 58–68 (2011)
29. De Schryver, C., Shcherbakov, I., Kienle, F., Wehn, N., Marxen, H., Kostiuk, A., Korn, R.: An energy efficient fpga accelerator for Monte Carlo option pricing with the Heston model. In: Reconfigurable computing and FPGAs (ReConFig), 2011 international conference. IEEE, Cancun, pp 468–474 (2011)
30. Ovtcharov, K., Ruwase, O., Kim, J.-Y., Fowers, J., Strauss, K., Chung, E.S.: Accelerating deep convolutional neural networks using specialized hardware. Microsoft Research Whitepaper **2**(11) (2015)
31. Sundararajan, P.: High performance computing using FPGAs. Xilinx white paper: FPGAs, pp 1–15 (2010)
32. Ouyang, J., Lin, S., Qi, W., Wang, Y., Yu, B., Jiang, S.: SDA: Software-defined accelerator for large-scale DNN systems. In: Hot chips 26 symposium (HCS), 2014 IEEE. IEEE, Cupertino, pp. 1–23 (2014)
33. Muslim, F.B., Ma, L., Roozmeh, M., Lavagno, L.: Efficient FPGA implementation of OpenCL high-performance computing applications via high-level synthesis. *IEEE Access* **5**, 2747–2762 (2017)
34. Coussy, P., Gajski, D.D., Meredith, M., Takach, A.: An introduction to high-level synthesis. *IEEE Des Test Comput* **26**(4), 8–17 (2009)
35. Ecoscale project. <http://www.ecoscale.eu/project-description.html>. Accessed 1 Nov 2018
36. Durand, Y., Carpenter, P.M., Adami, S., Bilas, A., Dutoit, D., Farcy, A., Gaydadjiev, G., Goodacre, J., Katevenis, M., Marazakis, M., et al.: Euroserver: energy efficient node for european microservers. In: Digital system design (DSD), 2014 17th Euromicro conference. IEEE, Verona, pp 206–213 (2014)
37. L. Struyf, S. De Beugher, D. H. Van Uytzel, F. Kanfers, T. Goedemé: The battle of the giants: a case study of GPU vs FPGA optimisation for real-time image processing. In: Proceedings PECCS, vol 1. VISIGRAPP **2014**, 112–119 (2014)
38. Scene 1.0—background subtraction and object tracking with TUIO. <http://scene.sourceforge.net/>. Accessed 1 Nov 2018
39. Lucas, B.D., Kanade, T., et al.: An iterative image registration technique with an application to stereo vision. In: Proceedings of DARPA Image Understanding Workshop, April 1981, pp 121–130 (1981)
40. Optical flow design example. <https://www.altera.com/support/support-resources/design-examples/design-software/opencl/optical-flow.html>. Accessed 1 Oct 2018
41. Bouguet, J.-Y.: Pyramidal implementation of the affine Lucas–Kanade feature tracker description of the algorithm. Intel Corp **5**(1–10), 4 (2001)
42. Yeshwanth, C., Sooraj, P.A., Sudhakaran, V., Raveendran, V.: Estimation of intersection traffic density on decentralized architectures with deep networks. In: Smart cities conference (ISC2), 2017 international. IEEE, Wuxi, pp 1–6 (2017)



Arslan Arif has done his masters from NUST Pakistan. Currently he is pursuing his PhD degree with Department of Electronics and Telecommunication (DET) Politecnico Di Torino, Italy. His current research interests include high-level synthesis (HLS), computation accelerators (FPGA and GPU) and internet of things (IoT)



Felipe A. Barrigon is a qualified Electrician, Mechanical and Electrical Engineer graduated at Carlos III University in Madrid (Spain). He currently works at Agustin de Betancourt Foundation as in-house researcher for ACCIONA Construcción Technology & Innovation Division, in the field of new technologies development for the civil engineering sector, e.g. development of automated systems for tunneling guidance and other underground construction works. He is currently involved in ECOSCALE project.



Luciano Lavagno received his PhD in EECS from U.C. Berkeley in 1992. He co-authored four books and over 200 scientific papers. He was the architect of the POLIS HW/SW co-design tool. Between 2003 and 2014, he was an architect of the Cadence Cto-Silicon high-level synthesis tool. Since 1993 he is a professor with Politecnico di Torino, Italy. His research interests include synthesis of asynchronous circuits, HW/SW co-design, high-level synthesis, and design tools for wireless sensor networks.



Francesco Gregoretti graduated in 1975 from Politecnico di Torino, Italy where is now a Professor in Microelectronics. From 1976 to 1977, he was an Assistant Professor at the Swiss Federal Institute of Technology in Lausanne (Switzerland) and from 1983 to 1985 Visiting Scientist at the Department of Computer Science of Carnegie Mellon University, Pittsburgh (USA). His main research interests have been in digital electronics, VLSI circuits, massively parallel multi-microprocessor

systems for VLSI CAD tools and in image processing architectures. More recently, his research has been focused to co-design methodologies for complex electronic systems, to methodologies for reduction of electromagnetic emissions and power consumption of processing architectures by the use of asynchronous methodologies.



Mihai Teodor Lazarescu Mihai Teodor Lazarescu received his PhD from Politecnico di Torino (Italy) in 1998. He was Senior Engineer at Cadence Design Systems, founded several start-ups and serves now as Assistant Professor at Politecnico di Torino. He coauthored more than 40 scientific publications and several books. His research interests include sensing and data processing for IoT, WSN platforms, and high-level hardware/software co-design and highlevel synthesis.



Javed Iqbal received the M.S. degree in telecommunications engineering from the Politecnico di Torino, Torino, Italy, in 2014, where he is currently pursuing the PhD degree with the Department of Electronics and Telecommunications. His current research interests include instrumentation and measurements, statistical signal processing, control systems, and the design and implementation of low-power sensors for indoor human detection, localization, tracking, and identification.



Liang Ma received the M.S. degree (with Hons.) from Politecnico di Torino, Italy, where he is currently pursuing the PhD degree with the Department of Electronics and Telecommunications under the supervision of Prof. L. Lavagno. His research interests focus on high-level synthesis, electronic system level design and low-power high-performance computing.



Manuel Palomino is a Telecommunication Engineer and Project Manager (PMP) within Acciona Technology and Innovation Division. He joined Acciona Construcción in 2009 and since then, he has been involved in several National and European R&D Projects related to ICT and Robotics such as MIROR, CABLEBOT, TITAM, MEGAROB and ECOSCALE.

Intelligence, SW, HW and embedded systems development, satellite integration, and High energy Physics. Main projects completed: HormiH: AI software for composite materials optimization, ENH: AI software for optimization of No. of tests needed for characterization of composite materials through multidimensional algorithms, BIOFER: System design for control of biological reactors, LOAD and EDIANA: Control and monitoring systems for fuel cells, solar panels and other energy management systems in buildings, ACCES: design of embedded RF-based guidance system for blind persons, MIROR: NDT system to automate testing of fiber composites, BLAST: Test system designed to test structures subjected to extreme TNT blasts to prevent terrorist attacks. He is currently involved in ECOSCALE project



Javier Luis Lopez Segura graduated in Physics at University of Granada (1988). He currently works at Acciona Construcción ICT research group. His research skills and interests include, among others, Parallel computing, Computer vision, Artificial