Postprint of: Czyżewski A., Bratoszewski P., Ciarkowski A., Cichowski J., Lisowski K., Szczodrak M., Szwoch G., Krawczyk H., Massive surveillance data processing with supercomputing cluster, Information Sciences, Vol. 296 (2015), pp. 322-344, DOI: 10.1016/j.ins.2014.11.013

Massive surveillance data processing with supercomputing cluster

A. Czyżewski[a], P. Bratoszewski[a], A. Ciarkowski[a,*], J. Cichowski[a], K. Lisowski[a], M. Szczodrak[a], G. Szwoch[a], H. Krawczyk[a]

[a] *Gdańsk University of Technology, Faculty of Electronics, Telecommunications and Informatics, ul. Narutowicza 11/12. 80-233 Gdańsk, Poland*

Abstract

In recent years, increasingly complex algorithms for automated analysis of surveillance data are being developed. The rapid growth in the number of monitoring installations and higher expectations of the quality parameters of the captured data result in an enormous computational cost of analyzing the massive volume of data. In this paper a new model of online processing of surveillance data streams is proposed, which assumes the use of services running within a supercomputer platform. The study presents some of the highly-parallelized algorithms for detecting safety-threatening events in high-resolution-video streams, which were developed during the research, and discusses their performance on the supercomputer platform.

Keywords: smart surveillance, massive video data processing, object detection, parallel processing

1. Introduction

Modern video surveillance systems are composed of a large number of cameras covering wide areas. It is difficult for the system operator to notice every important event. Many existing installations are based only on video recording for the post-factum offline

*Corresponding author

Email addresses: andcz@sound.eti.pg.gda.pl (A. Czyżewski), bratoszewski@sound.eti.pg.gda.pl (P. Bratoszewski), rabban@sound.eti.pg.gda.pl (A. Ciarkowski), jay@sound.eti.pg.gda.pl (J. Cichowski), lisowski@sound.eti.pg.gda.pl (K. Lisowski), szczodry@sound.eti.pg.gda.pl (M. Szczodrak), greg@sound.eti.pg.gda.pl (G. Szwoch), hkrawk@eti.pg.gda.pl (H. Krawczyk)

July 22, 2014

analysis. Automatic event detection improves the performance of the monitoring system by providing notifications on detected events that may potentially be security threats and that require the operator's attention [15]. Automatic analysis of video streams and event detection are subjects of many scientific studies [16]. However, analysis of even a single visual data stream with complex algorithms is computationally intensive. In case where multiple data streams, carrying various modalities, are analyzed, computational requirements increase substantially. Standard computer machines have limited resources and they are only able to analyze data from a small number of video sources, with a limited number of computationally-intensive algorithms. In practical applications, video streams from a large number of cameras have to be analyzed at the same time. Moreover, analysis of each video stream cannot be limited to one algorithm, because different types of complex analysis (e.g. object detection, people counting, intrusion detection, etc.) have to be performed at the same time in online mode. Achieving this goal on a typical computer machine is not possible. Additionally, if separate machines analyze individual video streams, there is a problem of integration of partial analysis results in order to perform a complex event detection in the whole observed area. In order to cope with these requirements it is necessary to employ an efficient platform equipped with adequate resources and capable of parallelizing the key computations.

In this article we address the problem of automated, near-realtime, analysis of massive surveillance data from multiple video cameras and using various complex analysis algorithms. We propose utilizing the computing power of a supercomputer platform for this task. Contrary to standard computer machines, a supercomputer platform provides resources (both the processing power and memory storage) required for concurrent and efficient processing of data from a large number of sensors in the surveillance system, employing a set of computationally complex algorithms. Single supercomputer nodes may be dedicated for processing single source data with a resource-demanding algorithm, while other nodes may integrate partial results from multiple sources running less complex algorithms. Such a system requires solving important problems related to algorithm implementation, data exchange, resource management, etc., but it also provides an efficient solution for the analysis of massive amounts of surveillance data. To the date, there have been no published works related to implementing a system for complex video

analysis and event detection on a supercomputer platform.

Implementation of the system described above on a supercomputer platform requires solving two important problems: implementation of individual algorithms on supercomputer nodes and efficient data exchange between the algorithms running on separate nodes. The latter issue was addressed within the project "MAYDAY EURO2012 – Supercomputer Platform for Context Analysis of Data Streams in Identification of Specified Objects or Hazardous Events", which proposes a new model of surveillance data processing, based on stream analysis services running within a dedicated hardware and software environment. Contrary to typical contemporary installations, the MAYDAY project not only deals with visual surveillance but also supports other modalities, like audio or thermovision, including fusion of these signals – multimodal analysis. Nevertheless, in this paper we focus primarily on the subject of video data processing and we refer to the literature regarding other aspects of the MAYDAY project (e.g. [28, 35, 36]). A low-level software layer for algorithm implementation and data exchange was realized within this project in a form of the KASKADA framework which utilizes the message transmission interface for data exchange between the nodes and provides an application development interface for the algorithm developers. Though implementation of this framework on the supercomputer platform lies outside the scope of this paper, some details may be found in the literature [30–32].

There are two main goals of this paper. The first of them is to demonstrate how a supercomputer platform may be utilized for processing massive data from the surveillance systems in an efficient way. For this purpose, the framework for multi-sensor data processing on a supercomputer platform is provided with some typical, computationally demanding algorithms for video analysis, example scenarios of video analysis in practical applications are presented and algorithms required for realizing these scenarios are described. It should be stressed that practical applications of the proposed system are not limited to utilizing the KASKADA framework and the algorithms presented here. Any framework which allows for an efficient data exchange between the supercomputer nodes may be used as the low-level layer and also other algorithms than presented here may be implemented within the system.

The second goal is to present how the most commonly used video analysis algorithms,

such as background subtraction or optical flow, may be efficiently implemented on multicore supercomputer nodes for parallel data processing. Efficient video processing requires a knowledge on computational resources needed by each algorithm. Supercomputer resources are expensive, so it is essential to assign the really needed resources to each algorithm only and to provide a mean to adjust them dynamically according to video features. For example, several low-resolution video streams may be processed on a single node, while the high definition video stream may need to be divided into two or more nodes in order to achieve a nearly realtime performance. Therefore, each of the individual video processing algorithms described here is implemented in the parallel form on a single supercomputer node and their performance in processing video streams of varying resolution is tested. These experiments provide knowledge on the number of processing cores needed for nearly realtime processing of a given video of a particular resolution by each individual algorithm.

The remainder of this paper is organized as follows. Section 2 briefly describes the general framework within which the video analysis algorithms are run. Section 3 is dedicated to details of each video processing algorithms and methods of their parallel implementation on the supercomputer node. The results of performance testing of each algorithm are presented and discussed in Section 4 and practical use cases of the proposed system are described in Section 5. The paper ends with conclusions and suggestions of the future research.

2. Framework for massive video analysis

Modern video surveillance systems are composed of a large number of cameras, many of which are of high resolution. Most of the algorithms that perform analysis of video streams for the detection of important events are computationally complex. This is most evident for algorithms that process the source camera images pixel-by-pixel, for example, a background subtraction performed for detection of moving objects. Therefore, the computation time increases with image resolution. A camera of 1920×1080 pixels providing images at 15 frames per second (fps) requires the algorithm to perform more than 31 million pixel operations per second. Therefore, for a sequential processing, video analysis in the online mode using a typical personal computer is possible only for low

to medium resolution streams and merging the analysis results from various sources and algorithms is usually problematic. In order to avoid such problems, a supercomputer platform is used to optimize the analysis by distributing the processing into multiple nodes that run the algorithms in parallel. Such a platform has to be able to process the data not only, but also to transfer the data among the algorithms and store them in the memory in an efficient way. In the case of 1920×1080 resolution cameras, an uncompressed (raw) image frame in 24 bits-per-pixel format uses more than 6 MiB of memory. Another advantage of using a supercomputer platform is the ability to run various, complex processing algorithms on separate computing nodes and to analyze a large number of video streams at the same time. Owing to that, the detection of some important events in a multi-camera surveillance system may be realized.

2.1. Related work

The proposed approach to processing of massive amount of surveillance data was developed in cooperation with the supercomputing section of Academic Computer Centre TASK, which allowed for virtually unconstrained use of the computing clusters available there. This opportunity greatly influenced the design of both the system, which was based on KASKADA platform developed by the TASK center employees, and consequently, the media processing algorithms. The mentioned KASKADA platform allows for manipulating, processing and storing multiplicity of multimedia (audio or video) streams. The access to such a powerful computing infrastructure put the authors in a very favorable, and simultaneously, quite uncommon position. To the Author's knowledge, no working solutions with capabilities comparable to the system described here existed earlier elsewhere. Therefore, it is not possible to compare the proposed solution with other supercomputing platforms capable to operate as a surveillance system part, so we will limit the comparisons to individual algorithms run on separate machines.

On the other hand, new emerging technologies related to Big Data paradigm and the idea of distributed processing of large data sets with clusters of commodity hardware allow for similar performance with a much lower cost, making it feasible to consider implementing proposed services upon such an architecture. Currently, the most notable platform enabling this kind of processing is Apache Hadoop framework [45]. Unfortunately, requirements related to data processing model implemented in Hadoop are quite

different to the requirements of the described approach. While Hadoop is primarily targeted at batch-oriented tasks and uses the distributed file system to share the data among the nodes, the surveillance algorithms are expected to work in quasi-realtime using online streams of data. Therefore, there is no straightforward way of porting them (or the supercomputing framework itself) to similar solutions, thus an additional research is needed to assess the feasibility of the process. On the other hand, some platforms exist which seem to be inherently better-suited to the task than those based on the MapReduce model. One notable example is the Storm distributed realtime computation system, which allows for processing of streams of realtime data in an arbitrarily complex way, through repartitioning the streams between each stage of the computation, much in the similar way as the KASKADA platform does [46], while not being tied to any particular hardware, as is in the Hadoop case.

Nevertheless, it is still possible to envision some ways which would allow to overcome the limitations of Hadoop, e.g., the streamed data could be partitioned into some smaller pieces and each of those pieces is distributed among the processing nodes as a separate file, which could be processed by the algorithms governed by the MapReduce scheduler. Unfortunately this leads to a suboptimal use of the computing resources due to several reasons. The analysis of multimedia streams typically requires the knowledge of the signal's past instances, so it is beneficial if the algorithm can run continuously, storing and maintaining its state in the fast memory. Running algorithms on independent, partitioned data requires the state to be shared among numerous, short-living processes. This also increases dramatically the overhead related to starting a new process with each portion of data, then reading out the state and then updating it after portion of data is processed.

The Storm-based solution supports the notion of continuous running an algorithm which maintains its state between subsequent portions of realtime data, but it has some other limitations. The ,,stream" abstraction in Storm has quite different meaning to the ,,multimedia stream" as defined in the KASKADA platform. In Storm's terms, streams relate to the data structured in low volume name-value tuples rather than in high volume, low-level binary data associated with the frames of digital multimedia signals and while it is possible to encapsulate such data in Storm tuples, additional serialization is required

which may render the solution ineffective.

The above described factors raise some important questions regarding these platforms and examining of these limitations is one of planned research topics by the authors'.

2.2. The supercomputer platform

An unprecedented volume of data acquired from surveillance system sensors calls for certain unconventional solutions. The approach proposed here is to use the supercomputer platform in order to deal with the task of online, near-realtime, analysis of numerous multimedia streams. To achieve this goal, two elements are necessary: the hardware – the supercomputing cluster, and the software, which would allow the processing load to be distributed among computing nodes, the multimedia streams to be acquired from the respective sensors, and then delivered to the analysis services, and finally: gathering the results.

A software layer implemented on a supercomputer platform is needed as the run-time environment and an enabler for streamlined development and management of multimedia processing services. The system described here was realized using the KASKADA software framework, although any software layer fulfilling the requirements stated here could be used. The KASKADA delivers essential interfaces providing the user with various functionalities as: running services with predefined parameters and manipulation of these parameters while the service is running. Moreover, previewing of the available media streams and the observation of results are possible, as well as the event handling. In order to deliver all these functions to the user, a layered model based on the client-server architecture had to be implemented (Fig. 1). Since the detailed architecture and the design of the KASKADA platform is outside the scope of this paper, the authors refer the interested reader to the literature where the overhead and implications related to the usage of KASKADA platform are discussed in detail [29–32]. The following paragraphs briefly focus on the services it offers enabling the massive surveillance data processing.

The lowest layer in the model presented in Fig. 1 constitutes the KASKADA platform which can be considered as a service server (with available media streams and services). To ensure a communication between the platform and the user application, the WebService mechanism based on the SOAP protocol was used. Consequently, the media streams

and services are described with XML structures and are managed with the SOAP protocol as well. Next, playing back and previewing of media streams, which were sent by user or were available in a media repository, occurs using RTSP protocol. The active services generate the output, as a result of processing the media stream. The results can be sent to the user via the event queuing system (basing on ActiveMQ), in the form XML structures. In order to integrate the user's application and the communication interface of the KASKADA platform, which is accessible via network protocols (HTML, SOAP, RTSP, and ActiveMQ), an abstraction layer was implemented on the client side. The abstraction layer is enclosed in a multi-platform library and enables the network protocols utilized to be transparent to the user.

In Fig. 1 the user's GUI application is described as GKliK (derived from the Polish acronym for Graphical KASKADA Client). It is based on mechanisms supported by the abstraction layer library libKliK. Fig. 2 shows an example of how the user's request is processed with respect to various layers of the system. The KliK application was developed in order to provide graphical user interfaces that are optimized and dedicated for particular services, because each service requires a different set of parameters. The graphical interfaces are prepared and tested by service developers. Hence running, tuning, and managing of services are facilitated in this way. Moreover, the KliK application interacts with the user, who can change and set parameters on the running service. Thus the restart of a service is unnecessary when the user wants to change parameters while the service is still working. An important advance of the GUI application is making it possible to control the application with either a keyboard or a mouse. The sse of the mouse is an important feature for services working on video images on which a ROI (Region of Interest) or other types of areas (e.g., barriers or points) can be marked as input parameters. The main task of the KliK application is a facilitation of user activities related to the operation offered by the KASKADA platform. As is said, the services work on the KASKADA platform which is implemented on the supercomputer machine. The client application can be considered as a kind of a bridge between the services user and the supercomputer environment. In order to realize this objective, a group of mechanisms needed to be implemented. The dependencies between particular functionalities are presented in Fig. 3.

3. Algorithms

This section focuses on discussing the algorithmic foundations of the massive surveillance data processing services developed by the authors. The following subsections will cover various approaches to analyzing the video streams assessed during the research (i.e. background subtraction and optical flow methods). In order to comply with ever-increasing consciousness of privacy protection rights this facet was investigated too, and is reflected in the research on privacy enhancement technologies.

3.1. Moving objects analysis

Detection of important events in surveillance cameras requires a number of processing operations to be performed. The main processing stages include: detection of moving objects, tracking of moving objects, classification of objects and detection of simple and complex events. The detection of moving objects is the most complex of all above procedures, because it operates directly on image pixels, and the processing time rises as the image resolution increases. However, most operations in the object detection stage may be performed independently for each pixel, so this algorithm is suitable for parallel implementation. The object tracking and event detection algorithms are much less complex. Therefore, this section of the article focuses mainly on parallel detection of moving objects in a video stream.

3.1.1. Detection of moving objects

The object detection algorithm analyzes the consecutive image frames from the camera. Its main task is to determine which image pixels belong to moving objects. The multi-stage procedure applied to that end is rather typical, so that it is easy to find algorithms descriptions in the rich literature covering the topic of surveillance video processing. There are three main stages of object detection [16]. Background subtraction separates pixels belonging to moving objects from those belonging to the static background. In most cases, the pixel values of the camera image are compared with a statistical model of the background. Additionally, object shadows are removed. As a result, a binary mask of foreground and background pixels is obtained. The second stage is morphological processing of the mask, which removes pixel noise and fills small gaps.

The last stage is segmentation, which extracts groups of foreground pixels that represent moving objects and describes them using contours. The background subtraction and shadow removal is the most time-consuming part of the algorithm. At the same time, each pixel may be processed independently from the others, so this procedure may be run in a parallel manner. In the case of morphological processing and object segmentation, parallel processing does not provide any substantial improvement in the processing time.

Various background subtraction algorithms were proposed in the literature. The Gaussian Mixtures Method [40] is one of the most popular solutions. Alternative approaches include a Codebook algorithm proposed by Kim [22]. For the purpose of the described event detector, it was decided to implement the Codebook algorithm, because it is more flexible in adapting to varying lighting conditions, it does not imply that background changes have Gaussian distribution, and no separate shadow removal is needed. The computational complexity of the Codebook and GMM algorithms were found to be comparable. However, the Codebook algorithms benefit from the parallel processing, because pixels in various image regions differ in the processing time, depending on the content variability [43].

In the Codebook method, each image pixel is represented in the background model with a number of codewords. A codeword is a vector:

$$\mathbf{c}_i = \langle \bar{R}_i, \bar{G}_i, \bar{B}_i, \check{I}_i, \hat{I}_i, f_i, \lambda_i, p_i, q_i \rangle \tag{1}$$

elements of which represent average color values $(\bar{R}_i, \bar{G}_i, \bar{B}_i)$ and minimal/maximal brightness values $(\check{I}_i, \hat{I}_i)$ represented by the codeword, a number of recent images in which the codeword was matched (f_i) and not matched (λ_i), the number of the frame in which the codeword was created (p_i) and the number of the frame for which the latest codeword update was performed (q_i).

The background subtraction is performed by checking whether the difference in both the color and the brightness between the current image pixel and any codeword in the background model keeps within the limits. A pixel (R, G, B) matches the codeword provided the brightness difference fulfills the condition:

$$\alpha \check{I} \leq \sqrt{R^2 + G^2 + B^2} \leq \min \left\{ \beta \check{I}, \frac{\hat{I}}{\alpha} \right\} \tag{2}$$

where α and β are the threshold values for shadows and highlights, respectively, that limit the range of pixel brightness changes due to lighting variations, and the color difference fulfills the condition:

$$\sqrt{(R^2+G^2+B^2)-\frac{\left(R\bar{R}+G\bar{G}+B\bar{B}\right)^2}{\bar{R}^2+\bar{G}^2+\bar{B}^2}} \leq \varepsilon \tag{3}$$

where ϵ is the threshold limiting the color variations.

If any codeword representing the pixel fulfills both conditions, this codeword is updated and the pixel is matched as a one belonging to the foreground. Values $\bar{R}, \bar{G}, \bar{B}$ are updated with the current pixel values R, G, B using a running average. The brightness range $(\check{I}_i, \hat{I}_i)$ may be extended, if needed. The statistical parameters f, λ, q are updated with the current frame number. If no matched codeword was found, the pixel is assigned to the background.

In the original Codebook algorithm [22] the background model is created during a dedicated training phase in which new codewords are added, if no matching codeword was found. After the training is finished, codewords that were not matched for a defined period are removed from the model. In the detection phase, the model is not updated. This approach has a serious drawback: variations in the lighting and the scene content are not included in the model and detection during the training phase is not possible. In order to avoid such problems, modifications of the algorithm, inspired by Kim's suggestions [23], were introduced. A three-layer background model is created. The permanent background is created during the training and is not modified in later stages. The long-term model layer consists of codewords added to the model during the detection phase. The third layer is a cache. Each new codeword is added first to the cache and is later either moved to the long-term layer or removed, depending on whether the codeword remains matched for a defined time. Additionally, the training process was optimized by not adding codewords that would be removed after the training because of the time constraints existence. Moreover, enabling object detection in the training phase is possible using a modified condition: a codeword that was matched and its number of consecutive non-matched frames lay below the half of the number of frames processed so far. This modification allows for a simplified object detection in the training phase with a satisfactory accuracy.

After the background subtraction, a binary mask denoting foreground and background pixels with 1 and 0 values, respectively, is obtained. The morphological processing [17] is used to remove noise with morphological opening (erosion followed by dilation) and then for filling small gaps in object contours with morphological closing (dilation followed by erosion). Both operations are performed using the 3×3 pixels square structuring element. The final operation is object segmentation. Connected components are extracted from the image of foreground pixels using the border-following algorithm [41]. Components with too small area are removed from further analysis. The remaining contours represent moving objects. They are described using a bounding box and they are used for further processing stages (object tracking and event detection).

3.1.2. Object tracking and event detection

The results of object detection are used for tracking the movement of each object and for detection of the selected events. The object tracking algorithm implemented in the described system is based on Kalman filters [16, 48]. Kalman trackers are used for prediction of the current position and size of the tracked objects. These estimates are then compared with the object detection results for tracker updating. The Authors' approach solves an important problem of resolving the ambiguous relations between the trackers and detected contours that occur when the objects obscure each other, split or merge, etc. [42].

The event detection is based on the analysis of the current and past states of each moving object, represented by the tracker. Each state describes the object features such as position, size, velocity, class, etc. The event detector examines how each object interacts with the scene (e.g., entering area defined in the camera view) and with other objects (e.g., leaving luggage). Each event is described using a rule, conditions of which allow for examining various object states and their changes in time. If the rule is fulfilled, an event is detected. In order to raise an alarm, the event has to be detected in a number of object states – this condition filters out short-term events and analysis errors [16]. Examples of basic events that are detected with this procedure include entering an area and crossing a barrier.

The detector for the basic events does not interpret the detected events – this is the task of a higher-level detector of complex events [16]. It allows for interpretation of the

type of object causing an event and time and spatial relationships between the detected events. For example, the basic event detector finds objects entering an area, while the high-level module checks whether the object is a car, if it stops in the area and remains there longer than for a defined limit, etc. Events detected by the high-level module may represent actual security threats (e.g., an unattended bag) and an alarm may be sent to the system operator. It should be noted that the object tracking and event detection algorithms have a relatively low computational complexity, therefore parallel processing is not necessary. However, parallel analysis of object detection results obtained from multiple sources allows for online event detection in multi-camera systems surveying large areas.

Fig. 4 presents an example of video analysis results using algorithms outlined in this chapter. The top left figure shows a binary mask, resulting from the background subtraction with the Codebook algorithm. Black pixels are those assigned to the background (with matching codewords in the background model). This mask is then cleaned with morphological operations in order to remove noise and fill small gaps (top right figure). Contours of moving objects are extracted in each frame and the position of each object in consecutive frames is determined using Kalman filters. The bottom left figure presents the current positions of the tracked objects (denoted with rectangles) and trajectories of their movement (lines). Finally, an event related to crossing a barrier (an exit gate from the parking lot) is detected when a trajectory of the object's movement crosses the barrier (bottom right figure). The object causing the event is marked on the screen and a relevant alarm message is sent to the operator.

3.1.3. Parallel algorithm implementation and complexity analysis

The background subtraction algorithm is the most computationally complex algorithm within the video analysis module. Therefore, a dedicated supercomputer node should be assigned to the task of object detection in a single video source. Object tracking and event detection are less complex, therefore a single node may run these algorithms on the background subtraction results obtained from several nodes and integrate the results from different sources. In the background subtraction algorithm based on the Codebook method, each pixel is processed separately. The processing is performed using two nested loops - first by image rows, then by pixels in the row. It is convenient to treat

a row of pixels as a unit assigned to the processing thread. A more fine granularity (e.g., by single pixels) is possible but is not expected to be beneficial. Rows of pixels may be assigned to threads either statically, before the processing (cutting the image into several parts and assigning each part to a separate thread) or dynamically (each thread receives a single row of pixels at a time; when it finishes processing, another row is assigned). Although the processing overhead is higher in the dynamic case, it is more suitable in the Codebook algorithm, since the number of codewords representing the pixel is not constant (it changes over time and it may be different for various pixels) and also it cannot be predicted which codeword will be matched. Therefore, the processing time of a single pixel may vary considerably, depending on the rate of image content changes. In stable image regions, the first codeword may be matched, while in some regions with intensive movement (e.g., a busy street), a large number of codewords may be analyzed and it is possible that none of them will match. Moreover, processing of each codeword may finish either after checking only one condition (brightness or color) or after checking both of them. A more detailed discussion of complexity of the parallel object detection with the Codebook algorithm may be found in an earlier publication [43].

Because of the above-mentioned issues, it is expected that the dynamic scheduling method will provide a better performance than the static one, especially if the changes of the image content are limited mainly to specific image sections. If the static method was used, some threads would be idle if they finished processing their part of the image before the others. The dynamic method ensures that the workload is balanced between the threads, at a cost of some overhead related to thread and resource management. In the experiments described in this article, the dynamic scheduling implemented in the OpenMP library [11] was utilized using the `#pragma omp parallel for schedule(dynamic)` statement in the outer processing loop.

3.2. Crowd behavior analysis

Gathering of a large number of people in a confined area may be the source of dangerous events. People attending concerts, sport games and other similar ceremonies may be exposed to serious physical injuries, and in the worst case they might even their lose lives. Emergency situations have occurred many times in history [18, 19, 44].

Crowding on pathways. One of the dangerous situations is a crush which may be caused by an obstructed pedestrian pathway. Obstruction of passages or exits may occur during crowded events, where many people are gathered in a small area. Such a situation may take place in a sports stadium or in an entertainment hall during a football game, a concert, etc. at the moment when everyone attending wants to leave the building at the same time. Regardless of the existence of multiple exits in the building, people tend to choose their preferred route, such as the way they entered the facility. This may result in a significant slowdown or the formation of blockages in places such as passages, halls near doors or elevators.

Various methods of video analysis of crowd behavior are undertaken by scientists. For example, Lo and Velastin describe a system that can serve for detection of overcrowding in underground station platform [34]. Overcrowding is obtained by subsequent main steps which consist of background subtraction and classification of obtained features like pixel count, energy, entropy by multilayer perceptron. In the last years, works aimed towards automatic detection of anomalies in human crowds have been proposed [3, 38].

In our approach, the state of pedestrian flow in a given area is determined by examining the rates of movement of pedestrians and their density, obtained by video analysis algorithms. Determination of the flow velocity of the crowd does not require detection and tracking of individuals, which would be cumbersome to implement. For this purpose a method based on optical flow combined with fuzzy logic is utilized. Average velocity of flow is calculated at a selected point in the frame, according to the people's direction of movement. Estimation of the state of congestion is performed by fuzzy logic. For example, if the observed velocity at checkpoints is decreasing and the occupancy of the area is large, blocking up of the area is probable. Automatic recognition of the degree of congestion at critical points allows the blockage to be detected and an appropriate response to be made, for example, by identifying alternative exit paths.

The processing of the video signal in nearly real time is required to provide a practically usable solution. Therefore, the supercomputer was employed to provide the hardware base for performing the calculations on the video streams. Parallelization is necessary due to the high computational complexity of the optical flow algorithm. For a single 704×576 video stream, up to 12 cores should be assigned, which means 1 node of the

applied supercomputer cluster.

People counting in crowds. Mass crowd gatherings such as sports games or concerts can be the source of various risks for individuals, particularly evoked by excessive numbers of people in specific places. Exceeding the value regarded as the safe limit may cause serious risks. The number of people in the enclosed area is obtained by video analysis algorithms, mainly because of their versatility. Such a counting is done in a place where people can enter the area, e.g., by locating a video camera near each entrance.

Various methods of counting people in crowded scenes have been recently used. Albiol et al. [2] describe a technique based on the analysis of the derivate image constructed from a time sampled section of original image. Another method [1] uses statistical analysis of object corners detected while people move. Both latter methods were investigated in some specific conditions of underground train doors. Bozzoli et al. [6] propose an approach based on the sparse optical flow method and pedestrians contours obtaining by edge extraction from the image. The proposed algorithm for counting people in the crowd differs from described approaches. It uses dense optical flow for motion analysis. In the input image, zones where people pass, e.g., through the door, are regarded as so-called "virtual gates". Each of them is processing images employing computationally complex optical flow methods. The algorithm is dealing with complex situations which, for example, may occur while people entering large public events [25]. Moreover, the algorithm is designed to work in a system with centralized architecture, where the video signals gathered from multiple cameras are being processed by the computer cluster.

3.2.1. Crowding detection

Crowding detection is realized in several stages. In the initial phase an optical flow field is determined to estimate the speed and direction of movement of pedestrians. The input for the algorithm performing this task is always two consecutive image frames. The next step is to analyze the traffic by calculating the average velocity of the flow of pedestrians in each of the previously defined checkpoints. The next step is to determine the occupation of the area by the method of image background subtraction. The obtained information is provided to the input of the fuzzy logic system, whose task is to give the final result of determining the state of congestion of the analyzed area. The concept of

the detector is illustrated in Fig. 5. The assumption is made that no other objects apart from people appear in the investigated area. Therefore, additional recognition of object type is not required. The control lines k_i are perpendicular to the exit path and defined separately for each camera. The average speed of people $v_{k,i}$ obtained by the optical flow method is determined for vectors crossing each control line. The speed vectors $v_{k,i}$ are calculated synchronously in each image frame.

Motion detection. The method based on calculation of the optical flow was utilized for detecting crowd motion speed and direction. The algorithm utilized for obtaining the optical flow field employs CLG (Combined Local Global) method [8]. Similarly to the characteristics of optical flow algorithms' coarse-to-fine strategy, this algorithm uses a multi-grid approach, where estimates of the flow are passed both up and down the hierarchy of approximations. The algorithm combines the advantages of the global Horn-Schunck approach [20] and the local Lucas–Kanade method [37]. Moreover, this was the best-performing algorithm according to the comparison study [5].

The CLG method computes the optical flow field $(u(x,y), v(x,y))^\intercal$ of the image sequence $f(x,y,t)$ at instant t by solving a system of the partial differential equations. The solution is found by the multi-grid methods [7]. The smallest density of the grid is determined by `max_level` parameter and the largest depends on the `start_level` parameter. The density of each intermediate grid between the most sparse and the most dense doubles.

Pedestrian flow analysis. The obtained continuous flow field (motion direction and velocity determined for each pixel) is sampled in fixed spatial density $(\Delta x, \Delta y)$. Vectors extracted in this way intersect with control lines. During processing of subsequent image frames $m-1$ and m, having defined the number of control lines K, we obtain sets of vectors representing instantaneous velocity $v_{m,k,i}$,
where: k – control line number, $k = 1...K$,
$i = 1...I$, I – number of vectors which intersect control line k.

The motion velocity in each control line k is calculated according to the following

equation:

$$v_{m,k} = \frac{1}{N_k} \sum_i v_{m,k,i} \tag{4}$$

where N_k is the number of vectors intersecting the control line. The final value of velocity v_k is found as a result of temporal averaging of speed (4) in a defined M frames period

$$v_k = \frac{1}{M} \sum_{m=1}^{M} v_{m,k} \tag{5}$$

The parameter z which represents occupancy of the area is obtained with the use of the background subtraction method [21] as defined in the equation:

$$z = \frac{P_{FG}}{P_{TOTAL}} \tag{6}$$

where P_{FG} – number of pixels not qualified as background, P_{TOTAL} – total number of pixels in image.

The fuzzy logic system is employed to make an assessment of the state of pedestrian flow [27, 49]. As determined in previous stages of processing, velocity and area occupancy parameters constitute the input data to the decision-making system. Mamadani's method was used as fuzzy inference technique. Membership functions defined for parameters v_k and z defined by equations (5) and (6), named `Speed{k}` and `Occupancy`, have triangle shape. For fuzzy `OR` and `AND` rules evaluation fuzzy union and intersection operators are applied. In defuzzification procedure, the centroid method is utilized.

The camera view and the corresponding video processing result are presented in Fig. 6.

Parallel algorithm implementation and complexity analysis. Calculation of optical flow vectors is the most computationally expensive element of image processing in the crowding detection algorithm. For the processing of a single video stream, a dedicated cluster node should be assigned to this task. The algorithm for calculating of optical flow is based on solving of elliptic partial differential equations involving the application of multigrid methods [7, 8]. The idea behind the multigrid is to use a sequence of coarse grids as a mean to accelerate the problem solving on the finest grid. Moreover, the result calculation of a new pixel value depends on its neighbors state. In this case a convenient method of parallel processing, based on an assignment of the incoming video frames to the individual threads, was utilized. An implementation method is related to a pool of

threads concept. An algorithm input requires a pair of subsequent image frames. Therefore, each pair of incoming frames is assigned dynamically to the processing thread. The number of threads in the pool is constant and it is one of the parallel program parameters. Considerable results are achieved on 24 threads per one supercomputer cluster node. Moreover, in order to increase speed of iterations through the image content, the Intel TBB (Threading Building Blocks) parallelization libraries were utilized.

3.2.2. Virtual gate algorithm

The Virtual Gate algorithm is based on the modified Optical Flow method. The method developed for counting people does not involve classifying modules because the aim of the algorithm is to detect the size and direction of motion of objects in video sequences having dimensions similar to the size of an average human body. Moreover, the Virtual Gate is used in places where human motion is expected, especially at entrances, passes, etc. Two parts of the algorithm can be distinguished: the main module which performs image processing and the calibration module. The Virtual Gate is devoted to counting people in a crowd passing through the scene observed by the camera. An illustration of the sample setup of the Virtual Gate is presented in Fig. 7A.

The detailed structure of the Virtual Gate is depicted in Fig. 7B. It is composed of a set of rectangular regions (R_i) situated next to each other and overlapping. The rectangles have identical shape and their size corresponds to the size of an average human body contour, with respect to its height and width (for a particular camera view).

The motion of objects is estimated in each region R_i using the Dense Optical Flow method [4, 9]. The set of vectors representing the direction and the velocity of the motion detected is obtained in the result of the above operation. Displacement vectors can be expressed by the planar vector field:

$$\mathbf{V} = V_1(x,d)\mathbf{i} + V_2(x,d)\mathbf{j} = V_\rho(x,d)\mathbf{e}_\rho + V_\varphi(x,d)\mathbf{e}_\varphi \tag{7}$$

where:

$\mathbf{i}, \mathbf{j}$ – unit vectors of x and d axes, $\mathbf{e}_\rho, \mathbf{e}_\varphi$ – unit vectors related to polar coordinates (the distance from the axis of symmetry, the angle measured counterclockwise from the positive x-axis).

Two directions of people's motion through the virtual gate are considered, namely forward and backward ("in" and "out", $+d$ and $-d$, as in Fig. 7). Moreover, a small divergence of the direction (α) is allowed because usually people do not maintain bearing while walking. The tolerance should not be too large, because of the need to discard those walking along the gate. Components of equation 7 can be written in a form of cylindrical (in this two dimensional case - polar) coordinates:

$$\begin{aligned} V_1 &= V_\rho cos(\varphi) + V_\varphi sin(\varphi) \\ V_2 &= V_\rho sin(\varphi) + V_\varphi cos(\varphi) \end{aligned} \tag{8}$$

Let φ_0 denote the angle corresponding to the direction pointed by d. New functions V_1^I, V_2^I, V_1^O, and V_2^O are calculated as given in equation 8 in order to obtain desired direction vectors (I-means "in", O-"out"):

$$\begin{aligned} V_k^I &= \begin{cases} V_k & \text{if } \varphi_2 \leqslant \varphi \leqslant \varphi_1 \\ 0 & \text{otherwise} \end{cases} \\ V_k^O &= \begin{cases} V_k & \text{if } \varphi_2 + \pi \leqslant \varphi \leqslant \varphi_1 + \pi \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{9}$$

where: $k = 1, 2$,

$\varphi_1 = \varphi_0 + \alpha$,

$\varphi_2 = \varphi_0 - \alpha$.

The number of origins of vectors directed towards "in" (L^I) or "out" (L^O), enclosed in each region R_i is obtained according to equation 10:

$$L^{(\bullet)} = \sum_{i=1}^{I} \sum_{j=1}^{J} \widetilde{V}_{i,j} \tag{10}$$

where:

$$\widetilde{V}_{i,j} = \begin{cases} 1 & \text{if } \left| \mathbf{V}_{i,j}^{(\bullet)} \right| > T_M \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

T_M – vector magnitude threshold,

$\mathbf{V}_{i,j}^{(\bullet)} = \mathbf{V}^{(\bullet)}(x_i, d_j)$ – vector with origin at (x_i, d_j), refer to equations 7, 8 and 9,

I, J – number of points in R_i along x and d axes, respectively.

The obtained L^I and L^O are compared with a threshold value T_S. The threshold is proportional to the average area of a human silhouette at a given camera view and is obtained experimentally. If $L^{(\bullet)}$ is greater than the threshold value T_S, the "in" or "out" people counter is incremented and rectangle R_i enters an inactive ("hold") state for the period of C frames. The inactive state means that R_i is not performing calculations. This operation is done in order to avoid counting errors and to allow the moving object to leave the area of R_i while not being counted more than once. The period of the "hold" state is obtained experimentally during the calibration.

The aim of the calibration is to find the optimal counting threshold (T) in order to improve the effectiveness of the algorithm. Searching is based on the bisection method which approaches the optimal result in sequential iterations. The error of counting is calculated in successive iterations as presented in equation (12).

$$\begin{gathered} S_0 = w_0 \cdot (y_0 - x_0) \\ S_i = w_i \cdot [(y_i - y_{i-1}) - (x_i - x_{i-1})] \\ S = \sum_{i=1}^{n} S_i \end{gathered} \tag{12}$$

where:

n – number of selected time moments,

$f_i = f_0, f_1, ..., f_n$ – frame number at selected time moment i,

$x_i = x_0, x_1, ..., x_n$ – real number of people passing through the gate at selected time moment i,

$w_i = w_0, w_1, ..., w_n$ – weight coefficient.

The variables are defined as follows:

$y_i = y_0, y_1, ..., y_n$ – number of people counted in the i-th selected time moment by the Virtual Gate algorithm,

$S_i = S_0, S_1, ..., S_n$ – counting error in the selected i-th time moment,

S – total counting error.

The weight coefficient is added in order to favor the passage of either individuals or

groups of people. In each step, partial errors and the total error are minimized and then the counting threshold is increased or decreased respectively according to equations (13) and (14). The calibration process stops when $T_{corr} \leq 1$.

$$T = \begin{cases} T + T_{corr}, & \text{if} \quad S > 0 \\ T - T_{corr}, & \text{if} \quad S \leq 0 \end{cases} \tag{13}$$

$$T_{corr} \leftarrow \frac{T_{corr}}{2} \tag{14}$$

where: T – counting threshold (value within range 0, 1...100), T_{corr} – counting threshold correction.

The results of practical application of the Virtual Gate algorithm in a large sports and entertainment hall are presented in Fig. 8.

3.3. Privacy protection

A large number of visual streams are able to transmit an extensive amount of data, part of which may be treated as sensitive personal data, posing a threat for social privacy. Because of video streams analysis regarding surveillance and security issues, preservation of personal privacy is of the utmost importance. The proposed solution allows each sensitive object detected in the processed stream to be protected. In some critical situations, there is the possibility to extract visual content without perceptual degradation, employing visual encryption algorithms.

The researches and results oriented on privacy enhancement are reported extensively in the literature. The state of the art approach realizes compression-independent reversible encryption [10] based on random permutation encryption, the protected objects inside a single video frame are encrypted with the same encryption key. The reversible anonymization method proposed by authors enables protecting different objects visible in the scene using unique encryption key, what brings opportunity to decrypt video stream partially. Nevertheless, the crypto-anonymization algorithm described in the section 3.3.2 bases on pixel relocation encryption [12], it is also compression-independent. Another state of the art privacy protection approach bases on digital watermarking [26], the sensitive data are stored in video stream as watermark. The disadvantage of the mentioned approach is fact that the lossy visual compression is the threat for embedded watermarks,

thus this type of processing has negative influence on hidden data. Another approach employs combined object tracking mechanism and bit stream encryption [50], the objects detection algorithm being oriented on people and their faces. The bit stream encryption is not as flexible as proposed by authors spatial domain encryption. The implemented by authors sensitive regions detection are focused on both faces and license plates detection, the principles of sensitive data detection and classification is clearly described in the section 3.3.1. In comparison to the state of the art, the proposed methodology implementations is also focused on an on-line massive surveillance data processing. The proposed data flow together with multicore architecture enable both encryption and decryption without latency and frame dropouts. Moreover, the high resolution streams are processed fluently and lossy visual compression poses no threat for protected data.

Nowadays, simple anonymization algorithms are used for privacy protection in the media broadcasting and in video surveillance. Several of these were implemented in the developed framework; however, each of them employs non-reversible data protection. A reversible approach to anonymization was also developed, as presented in the following subsections.

3.3.1. Sensitive objects detection

In order to enhance the privacy in video systems, the sensitive data for further anonymization must be defined. In the scope of this article three types of sensitive data in video images were assumed: moving objects (e.g., vehicles, persons' silhouettes), persons' faces and car plates. In order to perform anonymization of these data, appropriate detectors are introduced. A description of the moving objects detection algorithm is presented in the part 3.1 of this article. The face detection algorithm incorporates the method presented by Viola and Jones [47]. The car plate detection algorithm is based on searching the areas in the frame where the highest energy is concentrated. Energy in the image corresponds to the clusters of the edges found by performing convolution of the image frame with a Canny kernel. There are several works [24, 33, 39] in the literature about Viola and Jones face detection method and edge detection with parallel acceleration. Those topics are mainly focused on GPU implementations. Authors, despite the efforts did not manage to find any publications referring to the use of the supercomputers either in the field of face or plate detection. In the approach presented

by the authors, in order to benefit from utilizing the supercomputer platform and parallel processing, division of the input frame into overlapping subframes was introduced allowing the analysis of each subframe in separate thread. GPU approaches are similar in the manner of subdividing of the input frame into smaller regions (e.g., integral images of the input frame or individual pixels) and parallel analysis of those regions. However, the proposed method in comparison to GPU-based approaches is mostly oriented on concatenating data from high number of data streams at once rather than in parallelization of a single visual stream. Future work might concern the usage of supercomputers which are based on GPGPUs like the supercomputing system Titan [14] which utilizes hybrid architecture of both CPUs and GPUs. Future experiments might also concern searching for the faces and plates located only inside of the moving objects which are the outcome of processing block described in section 3.1 to minimize the rate of false positives. The results of detection are presented in Fig. 9. The coordinates and sizes of the detected faces or plates are then passed to the anonymization block, since they are treated as the regions of sensitive data.

The complexity of the proposed sensitive data detectors depends on the size of the input frame, number of the input visual streams, number of subframes to be analyzed and the chosen minimal and maximal sizes of objects to be localized. The more overlapping subframes will be chosen the more overhead will be generated due to deletion of duplicates found in the overlapping regions. The presented solution is scalable as every data stream can be analyzed by separate processing unit of the supercomputer simultaneously.

3.3.2. Anonymization of the visual streams

Non-reversible anonymization (simple). The methods of simple anonymization described in this paragraph allow for permanent destruction of information contained in the sensitive region. The majority of the existing techniques are commonly known from media and newspapers but until now they have been used only occasionally in order to make the sensitive material containing personal data available in the public domain. Applying the algorithms developed for object detection leads to automating the anonymization procedure and makes it possible to enhance privacy in a straightforward way. The methods described below are referred to as “simple”, meaning that the processing flow is fed forward and there are no procedures available for recovering the destroyed data.

- **Cutting out**: The easiest and the most trivial algorithm for privacy enhancement is based on erasing the ROIs content. The simple formula representing the algorithm kernel is given by (15).

$$I_{out}(x,y) = \begin{cases} I_{in}(x,y) & \rightarrow I_{mask}(x,y) \neq 0 \\ 0 & I_{mask}(x,y) = 0 \end{cases} \tag{15}$$

where:

  - I_{out} – output anonymized image;
  - I_{in} – input original image;
  - I_{mask} – binary image with sensitive object masks;
  - x – column number of modified pixel;
  - y – row number of modified pixel;

- **Blurring**: The presented algorithm realizes privacy protection by applying image filtering with a Gaussian low pass filter. The processing is more complicated than for the cutting out algorithm, but this approach also provides a sufficient level of privacy, and the image degradation is more acceptable for the viewer. The procedure of Gaussian blurring requires the convolution of the input image to be computed by employing a two-dimensional Gaussian kernel. The Gaussian blurring of the ROI is defined by (16).

$$I_{out}(x,y) = \begin{cases} I_{in}(x,y) & \\ I_{in}(x,y) * \left(\frac{1}{2\pi\sigma^2} \cdot e^{\frac{(x^2+y^2)}{2\sigma^2}} \right) & \rightarrow \begin{matrix} I_{mask}(x,y) \neq 0 \\ I_{mask}(x,y) = 0 \end{matrix} \end{cases} \tag{16}$$

where:

  - I_{out} – output anonymized image;
  - I_{in} – input original image;
  - I_{mask} – binary image with sensitive object masks;
  - x – column number of modified pixel;
  - y – row number of modified pixel;
  - σ – parameter defining standard deviation of Gaussian distribution;

Modifications of the standard deviation parameter σ influence image degradation. Obtaining different values of this parameter entails a different blurring kernel influencing the blur shape.

- **Mosaicing**: A more sophisticated algorithm is widely known as pixelization or a mosaic algorithm. First of all, the input image areas related with non-zero pixels from the binary mask have to be divided into non-overlapping blocks. The block size depends on the mosaic grain size, and a smaller grain leads to a lower image degradation and a weaker anonymization. The following steps require the mean pixel value of all pixels to be computed for each block. The last procedure fills whole blocks in the output image with the mean pixel value obtained in the previous step. The formula of the mosaic algorithm is given by (17).

$$I_{out}(x,y) = \begin{cases} I_{in}(x,y) \\ \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} I_{in}\left(\left\lfloor \frac{x}{m} \right\rfloor + i, \left\lfloor \frac{y}{n} \right\rfloor + j\right) \end{cases} \rightarrow \begin{matrix} I_{mask}(x,y) \neq 0 \\ I_{mask}(x,y) = 0 \end{matrix} \tag{17}$$

  where:

  - I_{out} – output anonymized image;
  - I_{in} – input original image;
  - I_{mask} – binary image with sensitive object masks;
  - x – column number of modified pixel;
  - y – row number of modified pixel;
  - m – number of columns in block (grain width);
  - n – number of rows in block (grain height);

- **Bit-Shifting**: The last algorithm developed by the authors realizes anonymization by applying image bit-shifting. The bit-shifting procedure is based on a low-level rolling operation. The pixels' components binary values are moved in such a way that MSB (Most Significant Bit) are replaced by LSB (Least Significant Bit). The shifting of pixels is defined by (18).

$$I_{out}(x,y) = \begin{cases} I_{in}(x,y) \\ I_{in}(x,y) \ll b \end{cases} \rightarrow \begin{matrix} I_{mask}(x,y) \neq 0 \\ I_{mask}(x,y) = 0 \end{matrix} \tag{18}$$

where:

- I_{out} – output anonymized image;
- I_{in} – input original image;
- I_{mask} – binary image with sensitive object masks;
- x – column number of modified pixel;
- y – row number of modified pixel;
- b – parameter defining number of shifted bits;

Modifications of the parameter b have an influence on image degradation. If the parameter value equals 0, no shifting is executed, while if the parameter equals 8, the shifting results are identical to cutting out. Obtaining different values of this parameter entails different image distortions.

Reversible anonymization (crypto-anonymization). The drawbacks arising from the naive approaches to protecting privacy-sensitive information have contributed to intensified research on reversible anonymization techniques. In particular, the naive approaches described in theprevious subsection caused irreversible image degradation, thus the sensitive information was irrevocably lost. The personal data protection must not cause any anonymizing of vandalism acts, crimes or other suspicious behavior. At the same time, personal information about accident witnesses could be as helpful as information about the perpetrator of an incident. Consequently, the novel algorithm for reversible anonymization was developed [12] and analyzed [13].

The pixel relocation algorithm operates in two-dimensional image representation and realizes an extremely efficient privacy protection despite the application of lossy compression. The idea of pixels relocation is based on pixels shuffling within the given area of interest, and assumes that only the pixels' positions are changed, while the pixels' components values remain the same as in the original image. A knowledge of relocation principles allows reversible shuffling to be executed, leading to recovery of the original content. The low computational complexity is achieved at the expense of incomplete resistance against lossy visual compression. The quality artifacts may be encountered with regard to the compression ratio. The length (number of schemes) of the relocation

cascade influences the processing time and the complexity of the encryption key. The application of the proposed methodology on a real image is shown in Fig. 10.

The image after anonymization does not contain any visible sensitive information, so data protection can be realized properly. The analysis of the channel requirements and the type of potential compression algorithm is necessary. The quality degradation may be encountered in the decrypted frames because of video compression.

4. Results

The experiments were performed using the "Galera" supercomputer, which is located in the premises of "TASK" Academic Computer Centre in Gdańsk. With its theoretical processing power of 50 TFLOPS it is a fast machine, consisting of 5376 processing cores (672 nodes connected with InfiniBand network) and a total of 25 TiB of system memory. During the course of the project, a new, dedicated supercomputer "Galera+" was built with a processing power of 21 TFLOPS, comprising 1152 cores (192 nodes). The algorithms described in this paper were implemented on the KASKADA framework installed on the supercomputer platform and their performance was tested for video streams with varying resolution. The results are presented and discussed in the following subsections.

4.1. Object detection algorithm

The video processing system consisting of object detection, object tracking and event detection algorithms was implemented on the supercomputer platform. Since the object detection stage is the most computationally intensive one, a separate cluster node was dedicated for background subtraction with the Codebook method, morphological processing and object extraction. The remaining algorithms (object tracking with Kalman filters and event detection) require much less processing resources, so a single node is sufficient for analysis of several video streams. However, for object detection, parallel processing of a single video source had to be implemented in order to achieve a satisfactory performance. The processing speed (in frames processed per second) was assessed during the tests for various source video resolutions, varying the number of processing cores employed and two strategies – the static and the dynamic one.

In the test described here, a video stream in three resolutions (640×480, 720×576, 1600×900 pixels) was processed using a single core and 2, 4, 6 and 8 physical cores. Each case was tested with the static and the dynamic scheduling method. Each test run was repeated 10 times and the results were averaged. The processing speed in frames per second for each case is presented in Table 1 and in Fig. 11.

Table 1: Results of evaluation of parallel object detection efficiency – number of frames processed per second

Threads	1	2	4	6	8
640×480, static	15.65	21.46	26.35	28.11	29.43
640×480, dynamic	15.73	27.63	41.84	49.99	53.22
720×576, static	11.19	15.63	23.97	29.93	33.74
720×576, dynamic	11.10	18.62	29.83	36.44	40.26
1600×900, static	2.73	2.52	3.64	4.64	5.26
1600×900, dynamic	2.78	3.20	5.13	6.57	7.30

Object movement in test videos was not uniform in the camera view, so it was expected that the static scheduling method would be slower than the dynamic one, because threads related to processing of image parts with little or no movement are expected to finish their work earlier and will wait for threads that process image parts containing more moving objects. This was confirmed in the tests, as the dynamic scheduling performed better for all resolutions. In the dynamic method, single rows of pixels were assigned to each thread at a time. In other tests, groups of pixel rows were assigned to threads, but this did not improve the algorithm performance.

The source fps of test videos was equal to 15. It was assumed that for online video analysis, the processing fps should be at least twice the source fps, in order to take the platform overhead (related to video encoding/decoding, copying data, resources management, etc.) into account. It may be observed that for lower video resolutions and the dynamic scheduling, four processing threads are enough. For medium resolutions, six processing cores provide satisfactory performance. However, even 8 cores used for processing the high resolution video stream were not sufficient for online analysis. Increasing the number of threads results in an improved performance for all video resolutions. How-

ever, increasing this number beyond 6 cores does not yield a substantial improvement. In order to achieve online processing of high resolution videos, more than one supercomputer node has to be used. Dedicated algorithms are needed for splitting the image into parts, sending each part to the object detection algorithm working on a separate node and merging the results. In this approach, synchronization issues have to be taken into account. On the basis of the experiments it was decided that each instance of the parallel object detection algorithm running on a dedicated node and processing either the complete image or a section of a high resolution image, should use 6 processing threads.

Relating the results obtained for a supercomputer node to a single machine performing the same computations, a computer equipped with a powerful CPU with at least 4 processing cores should have sufficient power to process low resolution videos in online mode, taking into account that it also has to decode the video and to perform other pre- and post-processing tasks, which on the supercomputer are done by other nodes. For medium to high resolutions, the processing power of a separate machine is too low and the video stream would have to be down-scaled in order to achieve online processing feature, at a cost of information loss, however.

4.2. Crowding detection results

The algorithm for crowding detection was implemented in the form of KASKADA complex service. The most computationally complex element of the crowding detection algorithm related to calculation of the optical flow requires solving of elliptic partial differential equations [8]. Obtaining a solution with multigrid methods is performed by applying recursive algorithms such as V-cycle or W-cycle for traversing between coarse and fine grids [7]. The parallelization method based on a thread pool, which assigns the incoming video frames to the individual threads, was implemented. The performance of the crowding detection algorithm was described by the popular measure, namely the number of frames processed in one second – fps. The aim of the experiments was to determine the effective fps of the output stream. The experiments were performed for 704×576 and 1920×1080 video streams, various number of threads and the `start_level` algorithm parameter. The results of measurements conducted on a single node of the KASKADA platform (cluster Galera) containing 12 cores are shown in Table 2.

Table 2: Performance of image processing (fps) for one node of Galera cluster

Video size	704×576		1920×1080	
	`start_level`		`start_level`	
threads	0	1	0	1
8	6.74	19.97	-	-
12	8.32	23.62	0.25	1.15
24	8.09	24.39	0.38	1.72

The presented measurement results show that one node of the cluster is needed for processing of one 704×576 stream. Assuming that the video streams originating from a monitoring system in a large building are analyzed, the use of the supercomputer is thereby justified. Moreover, in such an approach the computational resources are used efficiently. High resolution video requires processing on 8 nodes in order to achieve satisfactory output frame rate. The value of the parameter `start_level` has a significant influence on the processing time, particularly in the case of 0, which means that calculations are performed on the image of original resolution (not scaled down).

4.3. Privacy protection results

The practical application of the presented privacy enhancement methodology was implemented as a complex service in the KASKADA framework [30]. Sample anonymized frames are shown in Fig. 12. The most common object classes related to surveillance systems are faces (Fig. 12a), license plates (Fig. 12b) and whole silhouettes (Fig. 12c). The decrypted frames after anonymization can not be published due to privacy issues.

The main disadvantage of the presented algorithm, apart from dependence on the type of color space, is the processing time. There are three factors affecting the processing time and its cost. A variety of presets was simulated and measured according to different key word lengths, various sizes of ROI and different numbers of areas. The experiments were performed on a desktop computer, on a single node of a supercomputer and on the node group of a computer cluster. The desktop test bed was based on the Intel Core i5-650 CPU (2 cores, 4 threads, 3.2 GHz) supported by 8 GiB random access memory.

Cluster nodes were based on the Intel Xenon Core E5345 CPUs (8 cores, 16 threads, 2.33 GHz), together with at least 16 GiB RAM.

4.3.1. Simulations focused on encryption key complexity

Firstly, the influence of the encryption key complexity on the processing time was measured. The encryption keys were designed to realize 1, 2, 4, 8, 16, 32, 48, 64, 96, 128, 192, 256, 384, 512 and 640 iterations. The processed ROI size was heuristically defined as 100×100, (the commonly encountered size of the detected faces in high resolution surveillance streams). The processing time for various key complexities and different test environments is presented in Fig. 13.

The processing time for a single node (plain gray line) was equal to that for the cluster (dotted black line). On the other hand, the processing took less time on a desktop PC (dashed black line) than on two other systems. The node/cluster computations were prepared for parallel processing, thus tasks were organized into separated threads, and after algorithm execution the thread supervising mechanism took place [30]. The management thread generates a computational overhead, so processing time is likely to increase.

4.3.2. Simulations focused on ROI size

The following part of the experiment concerns the influence of the encrypted area dimensions on the processing time. Employing the results obtained in the first simulation, it is possible to design an adequate encryption key realizing 48 relocation procedures. The test set contained objects with dimensions 32×32, 48×48, 64×64, 96×96, 128×128, 192×192, 256×256, 320×240 (QVGA), 640×480 (VGA), 768×576 (PAL), 800×600 (SVGA), 1024×768 (XGA), 1280×720 (HD720), 1280×1024 (SXGA), 1920×1080 (HD1080). The square root of the total number of pixels inside the specific area of interests is given on the x axis to clarify the results presented in Fig. 14.

At the beginning, the processing time on a desktop unit (dashed black line) was lower than on the node/cluster (plain gray/dotted black line), and the computational overhead for threads management was greater than for algorithm execution. The processing time of an object with dimensions 192×192 pixels was the same for each test environment. The colored lines intersect and the crossing point's time value is equal to 120 ms. Processing

of objects greater than 192 × 192 pixels was faster on the node/cluster than on the computer, despite the fact that the clock frequency was greater for the PC processor. In the border conditions, the difference of processing time between the desktop and the node/cluster is 1400 ms.

4.3.3. Simulations focused on a number of objects

Finally, the occurrence of a large number of objects was simulated employing the existing supercomputing environment. The test sets contained different numbers of manually defined objects in the visual stream, with: 1, 2, 4, 8, 12, 16, 24, 32, 48, 64, 96, 128 or 160 non-overlapped objects. The object size was fixed at 96 × 96 pixels, the encryption key was set to perform 48 relocation cycles, and test constraints were appointed using the previously obtained results. Each cluster node is able to compute the results employing up to 16 parallel threads. The results of the experiments are plotted in Fig. 15.

Processing up to four objects was faster on a desktop PC than on each other setup. If more than four objects were encountered in the processed frame, then computation on a node/cluster is faster than on the desktop computer. The processing time on a PC increases linearly. Processing on a node consumes the same amount of time as on the whole cluster up to the moment when 16 objects appear in the video frame. For more than 16 objects, processing was distributed among an adequate number of nodes. The node processing time increases non-linearly together with the increasing number of objects. The differences between processing time on a PC and on the node is 2000 ms for the maximum number of objects. The processing time for the cluster computation achieves saturation if more than 16 objects are simultaneously processed. Further increasing number of objects did not influence the processing time which remained constant for more than 16 objects. Parallel processing on the cluster was thereby realized, whereas for a single node and for more nodes, the processing time is 440 ms. Consequently, it turned out that processing of the video stream with the 160 objects requires the use of 10 nodes of the supercomputer cluster.

4.3.4. Complexity assessment

The performed experiments prove that only a single factor influences notably the computation time. The processing time increases linearly with increasing number of the

sensitive objects. To solve the problem with algorithm efficiency and on-line processing the encryption procedure for each object is realized simultaneously on the different cluster nodes. The parallelization of the anonymization procedure enables encrypting the sensitive objects employing a unique key for each object. The large number of objects which have to be anonymized is the main stress factor for proposed privacy enhancement solution. The processing of visual data from crowded places generates huge amount of sensitive data represented by sensitive objects. The parallel processing is the way for on-line visual streams protection.

5. Processing of massive data – a use case

In the previous sections, parallel implementation of important video processing algorithms and the assessment of their performance were presented. As it may be concluded, computationally intensive processing of a single, high-resolution video source with algorithms such as object detection or the optical flow, requires assigning a whole supercomputer node for the task of analyzing one data source with a single algorithm in order to achieve nearly-realtime data analysis. In practical situations, multiple data sources (a large number of surveillance cameras) have to be analyzed at the same time, and each source may be processed employing several algorithms. With the help of a supercomputer platform, all these sources may be analyzed concurrently by algorithms running on separate nodes, and the results may be integrated. With this approach, processing resources may be managed optimally and the complexity of a distributed network of separate video analyzing machines and the result integrating servers are avoided.

As an example of utilizing the supercomputer platform for massive data processing, a hypothetical surveillance system installed in a sport venue will be discussed. The system may consist of a large number (let's assume 100) of video cameras. Some cameras monitor entrances and exits of the building. Data from these cameras are analyzed by the virtual gate algorithm and the results (person count) are sent to the detector. At the same time, other cameras monitor the corridors and the video stream from each of these cameras is analyzed by the optical flow algorithm (which requires a whole node), providing data on the direction of people movement and the detected congestion cases. Some other cameras monitor off-limits areas. The object detection algorithm is run on a single node per

camera, while the object tracking and low-level event detection integrates several sources on a node, informing the system of potential intrusions. All these partial results are integrated by the final, high-level event detector which analyzes the results from all the algorithms processing different sources, and produces alarms informing of area intrusions, congestion on some corridors or at entrances, etc. Simultaneously, the anonymization algorithm is run for every camera in order to hide the privacy-sensitive data for the task of video presentation. The supercomputer platform KASKADA manages the whole system, providing the data for analysis by all the algorithms, distributing resources, exchanging data between algorithms and processing the detected events.

5.1. Advantage of the supercomputing computation

The main motivation for an implementation of the presented algorithms on a supercomputer cluster outside the scientific incentives was the fact that only especially designed architecture enables to achieve the sufficient efficiency with regards to massive data processing challenge. The processing time for presented algorithms was measured and extrapolated for low definition visual streams Table 3, for high definition visual streams Table 4. The specific hardware setups were analyzed, the numbers of processor cores/threads involved for the visual stream processing were considered.

Table 3: Processing framerates extrapolation for the presented algorithms for low definition video streams

Algorithms processing framerates PAL 704×576							
Number of threads	1	2	4	8	12	16	24
Optical flow FPS	2.04	3.37	5.16	16.49	19.04	21.98	30.03
Background subtraction FPS	21.96	27.65	43.87	65.05	85.43	112.20	147.36
Face detection FPS	20.56	23.11	29.65	38.05	45.60	53.46	56.58
Encryption FPS	14.16	14.55	15.41	17.46	20.14	23.80	37.36
PC processing (cascade) FPS	1.53	2.25	3.17	6.27	-	-	-
Cluster processing (parallel) FPS	-	-	-	-	-	-	30.03

In order to justify the approach described in this article the two scenarios were analyzed. The first four rows collected in the tables represent the processing framerates for

Table 4: Processing framerates extrapolation for the presented algorithms for high definition video streams

Algorithms processing framerates HD 1920×1080							
Number of threads	1	2	4	8	12	16	24
Optical flow FPS	0.48	0.98	1.45	2.65	4.91	5.93	9.05
Background subtraction FPS	3.84	5.27	8.55	13.08	15.83	15.83	15.83
Face detection FPS	9.12	9.83	11.49	13.42	15.05	16.65	18.67
Encryption FPS	9.49	9.72	10.22	11.38	12.83	14.72	20.82
PC processing (cascade) FPS	0.39	0.71	1.01	1.62	-	-	-
Cluster processing (parallel) FPS	-	-	-	-	-	-	9.05

the separately running algorithms. The fifth row represents an extrapolation of processing framerate considering the usage of a desktop computer; in this case algorithms were working in a cascade. The computer resources (cores/threads) processing framerate was selected from 1.5 to 6.3 FPS for PAL visual stream and from 0.4 to 1.6 FPS for HD visual stream. The sixth row is an extrapolation of the processing framerate obtained using a single node of the supercomputer cluster; in this case the processing was parallelized and the resultant framerate was similar to the framerate of the most complex algorithm. The ratio of the processing time using the desktop computer to the single node of the supercomputer cluster is higher than 5 and it is practically independent on visual stream resolution. The results are presented in form of columns chart in Fig. 16 for low resolutions, and in Fig. 17 for high definition.

The KASKADA supercomputing framework enables processing massive data with nearly-realtime performance, there is a limitation related to the number of available nodes. Practically the Galera supercomputer resources are sufficient to analyze the huge amount of data obtained with high definition cameras and another sensors located around the urban area.

6. Conclusions

A novel approach to the analysis of massive video data employing the supercomputer platform was presented. A selected set of computationally complex, but widely used video analysis algorithms, as well as performance of their parallel implementation were evaluated and discussed. It may be concluded that using a standalone machine would limit the processing capabilities to a small number of cameras and algorithms, thus leading to the necessity of employing of a complex, distributed client-server system. The main advantage of the approach proposed by the authors is the ability to analyze a large number of video streams processed by complex algorithms, at the same time, using a unified supercomputer system, with optimal resource management.

The scenarios presented in the paper utilize typical computationally intensive video analysis algorithms, especially demanding resources for high resolution images. Thanks to the parallel implementation of algorithms on the supercomputer platform, low and medium resolution video streams from cameras may be analyzed in nearly realtime. Using the supercomputer platform allows for a concurrent video event detection in a multi-camera system. One of the most computation power demanding algorithms is the object detection with background subtraction. During the tests it was observed that running separate instances of this algorithm on different supercomputer nodes allows for a concurrent processing of video streams of low to medium resolution. However, a single supercomputer node was not sufficient to process high resolution video streams. The supercomputer platform may be useful in this case for distributing the task of processing single high-resolution frames among several processing nodes.

The algorithm for online detection of people crowding in passages near sensitive areas has an impact on the quality of pedestrian service in large public buildings. Numerous points where a blockage can occur in e.g., a large concert hall induce a large number of input streams to be analyzed simultaneously. Multi-camera input streams should be analyzed online, which requires the parallelization. Assuming that the video streams originating from a monitoring system in a large building are analyzed, the use of a supercomputer is thereby justified. Moreover, in such an approach the computational resources are used efficiently. In another scenario, online counting of people in crowds employing the analysis of motion data obtained with the dense optical flow estimated

through the devised Virtual Gate algorithm was implemented. Similarly to the previous case, the number of input streams is high, for a large sports and entertainment hall with the target people counter installation.

The online reversible anonymization protects privacy without compromising of safety. High computational cost of encryption is a problem in a real life implementation. The problem has been solved using the supercomputer cluster for cryptographic anonymization. The developed algorithmic background allows a balance to be achieved between security and privacy issues in smart surveillance systems. The presented concept and observations resulted from real implementation of the algorithm in the supercomputer framework including services for enhancing privacy and anonymity preserving in a multi-camera surveillance system.

An example of using the proposed approach in a real-life scenario was also presented. With the same approach, other video analysis algorithms may be implemented in a parallel manner and added to the system repository. In this way, a flexible system for building the processing chains realizing different scenarios, according to the needs of the platform clients, was created. Therefore, the approach presented by the authors utilizes processing capabilities of a supercomputer platform for the task of improving the public security.

The proposed system was realized using the specific supercomputer platform and the developed processing framework. The reason for this was that no similar solutions, fulfilling the requirements described here, existed at the time this research was done. However, this does not implicate that the results can not be used outside the platform employed in the experiments. Similar supercomputer platforms may be constructed in the future and the general idea of massive data processing presented here, as well as results of performance testing performed for the algorithms described in this paper, will remain valid. Moreover, solutions alternative to standard supercomputer systems may become common in the near future. Particularly, the Storm system which allows for a concurrent analysis of multiple streams of data with various algorithms without using a specific hardware platform seems to be a particularly promising one, provided effective means of multimedia data serialization are implemented. Authors believe that the results presented here will be helpful for the scientists involved with complex, concurrent analysis

of multiple video streams on the future supercomputing platforms.

Acknowledgment

This research was funded within the project No. POIG.02.03.03-00-008/08, entitled "MAYDAY EURO 2012 – Supercomputer Platform for Context Analysis of Data Streams in Identification of Specified Objects or Hazardous Events". The project is subsidized by the European Regional Development Fund and by the Polish State budget.

References

[1] A. Albiol, A. Albiol, J. Silla, Statistical video analysis for crowds counting, in: Image Processing (ICIP), 2009 16th IEEE International Conference on, pp. 2569–2572.

[2] A. Albiol, I. Mora, V. Naranjo, Real-time high density people counter using morphological tools, IEEE Transactions on Intelligent Transportation Systems 2 (2001) 204–218.

[3] S. Ali, M. Shah, Floor fields for tracking in high density crowd scenes, in: D. Forsyth, P. Torr, A. Zisserman (Eds.), Computer Vision – ECCV 2008, volume 5303 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2008, pp. 1–14.

[4] B. Atcheson, W. Heidrich, I. Ihrke, An evaluation of optical flow algorithms for background oriented schlieren imaging, in: Experiments in Fluids, volume 30, pp. 467–476.

[5] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, R. Szeliski, A database and evaluation methodology for optical flow, in: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, pp. 1–8.

[6] M. Bozzoli, L. Cinque, E. Sangineto, A statistical method for people counting in crowded environments, in: Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference on, pp. 506–511.

[7] W. Briggs, V. Henson, S. McCormick, A Multigrid Tutorial, SIAM Books, Philadelphia, 2000. Second edition.

[8] A. Bruhn, J. Weickert, C. Schnorr, Combining the advantages of local and global optic flow methods, Combining the Advantages of Local and Global Optic Flow Methods, volume 2449 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, 2002, pp. 454–462. 10.1007/3-540-45783-6_55.

[9] A.J. Burt, E.H. Adelson, The laplacian pyramid as a compact image code, in: IEEE Trans. Commun., volume 31, pp. 532–540.

[10] P. Carrillo, H. Kalva, S. Magliveras, Compression independent reversible encryption for privacy in video surveillance, EURASIP Journal on Information Security 2009 (2009) 429–581.

[11] B. Chapman, G. Jost, R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming, The MIT Press, 2007.

[12] J. Cichowski, A. Czyżewski, Reversible video stream anonymization for video surveillance systems based on pixels relocation and watermarking, in: IEEE 13th International Conference on Computer Vision Workshops (ICCV2011), Workshop on Visual Surveillance, Barcelona, Spain, pp. 1971–1977.

[13] J. Cichowski, A. Czyżewski, B. Kostek, Visual data encryption for privacy enhancement in surveillance systems, in: J. Blanc-Talon, A. Kasinski, W. Philips, D. Popescu, P. Scheunders (Eds.), Advanced Concepts for Intelligent Vision Systems, volume 8192 of *Lecture Notes in Computer Science*, Springer International Publishing, 2013, pp. 13–24.

[14] Cray, Supercomputing system Titan, `http://www.olcf.ornl.gov/titan`, 2012. [Online; accessed 2013-12-02].

[15] A. Czyżewski, A. Ciarkowski, P. Dalka, P. Szczuko, G. Szwoch, P. Żwan, W. Jędruch, P. Kozielecki, Multimedialny system wspomagający identyfikację i zwalczanie przestępczości oraz terroryzmu (in polish), Multimedialny system wspomagający identyfikację i zwalczanie przestępczości oraz terroryzmu (in Polish), Wolters Kluwer Polska, 2011, pp. 211–227.

[16] A. Czyżewski, G. Szwoch, P. Dalka, P. Szczuko, A. Ciarkowski, D. Ellwart, T. Merta, K. Łopatka, L. Kulasek, J. Wolski, Multi-stage video analysis framework, Multi-stage video analysis framework, Intech, 2011, pp. 147–172.

[17] E.R. Dougherty, R.A. Lotufo, Hands-on Morphological Image Processing, SPIE Press, 2003.

[18] W.L. Grosshandler, N. Bryner, D. Madrzykowski, K. Kuntz, Report of the Technical Investigation of The Station Nightclub Fire, NIST NCSTAR 2, National Institute of Standards and Technology, Gaithersburg, Maryland, 2005.

[19] D. Helbing, A. Johansson, H.Z. Al-Abideen, Dynamics of crowd disasters: An empirical study, Phys. Rev. E 75 (2007) 046109.

[20] B. Horn, B. Schunck, Determining optical-flow, Artificial Intelligence 17 (1981) 185–203.

[21] P. Kaewtrakulpong, R. Bowden, An improved adaptive background mixture model for realtime tracking with shadow detection, in: Proc. 2nd European Workshop on Advanced Video Based Surveillance Systems, AVBS01, VIDEO BASED SURVEILLANCE SYSTEMS: Computer Vision and Distributed Processing, Kluwer Academic Publishers, 2001.

[22] K. Kim, T. Chalidabhongse, D. Harwood, L. Davis, Real-time foreground-background segmentation using codebook model, Real-Time Imaging 11 (2005) 172–185.

[23] K. Kim, D. Harwood, L. Davis, Background updating for visual surveillance, in: Proceedings of the International Symposium on Visual Computing, pp. 1–337.

[24] J. Kong, Y. Deng, GPU accelerated face detection, in: 2010 International Conference on Intelligent Control and Information Processing (ICICIP), Dalian, China, pp. 584–588.

[25] K. Kopaczewski, M. Szczodrak, A. Czyżewski, H. Krawczyk, A method for counting people attending large public events, Multimedia Tools and Applications (2013) 1–13.

[26] P. Korus, W. Szmuc, A. Dziech, A scheme for censorship of sensitive image content with high-quality reconstruction ability, in: Multimedia and Expo (ICME), 2010 IEEE International Conference on, pp. 1073–1078.

[27] B. Kosko, Fuzzy engineering, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1997.

[28] J. Kotus, K. Łopatka, A. Czyżewski, Detection and localization of selected acoustic events in 3D acoustic field for smart surveillance applications, Multimedia Tools and Applications (2012).

[29] H. Krawczyk, K. Bańczyk, J. Proficz, Parallel processing of multimedia streams, Computer Applications in Electrical Engineering 8 (2010) 9–25.

[30] H. Krawczyk, R. Knopa, J. Proficz, Basic management strategies on KASKADA platform, in: EUROCON - International Conference on Computer as a Tool (EUROCON), 2011 IEEE, pp. 1 – 4.

[31] H. Krawczyk, J. Proficz, KASKADA — multimedia processing platform architecture, in: International Conference on Signal Processing and Multimedia Applications, SIGMAP 2010.

[32] H. Krawczyk, J. Proficz, Real-time multimedia stream data processing in a supercomputer environment, Real-time multimedia stream data processing in a supercomputer environment, InTech, 2012, pp. 289–312.

[33] E. Li, B. Wang, L. Yang, Y. Peng, Y. Du, Y. Zhang, Y. Chiu, GPU and CPU cooperative accelaration for face detection on modern processors, in: 2012 IEEE International Conference on Multimedia and Expo (ICME), Melbourne, Australia, pp. 769–775.

[34] B. Lo, S. Velastin, Automatic congestion detection system for underground platforms, in: Intelligent Multimedia, Video and Speech Processing, 2001. Proceedings of 2001 International Symposium on, pp. 158–161.

[35] K. Łopatka, A. Czyżewski, H. Krawczyk, Automatic recognition of events in audio data using supercomputer cluster, in: 130th Convention of the AES.

[36] K. Łopatka, J. Kotus, A. Czyżewski, Application of vector sensors to acoustic surveillance of a public interior space, Archives of Acoustics 36 (2011) 851–860.

[37] B. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI '81), pp. 674–679.

[38] R. Mehran, A. Oyama, M. Shah, Abnormal crowd behavior detection using social force model, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pp. 935–942.

[39] S. Niu, J. Yang, S. Wang, G. Chen, Improvement and parallel implementation of canny edge detection algorithm based on gpu, in: International Conference on ASIC (ASICON), Xiamen, China, pp. 641–644.

[40] C. Stauffer, W.E. Grimson, Adaptive background mixture models for real-time tracking, in: Proc. of IEEE Conference on Computer Vision Pattern Recognition, pp. 246–252.

[41] S. Suzuki, K. Abe, Topological structural analysis of digitized binary images by border following, CVGIP 30 (1985) 32–46.

[42] G. Szwoch, P. Dalka, Resolving conflicts in object tracking for automatic detection of events in video, Elektronika 55 (2010) 520–525.

[43] G. Szwoch, D. Ellwart, A. Czyzewski, Parallel implementation of background subtraction algorithms for real-time video processing on a supercomputer platform, Journal of Real-Time Image Processing (2012).

[44] P. Taylor, The Hillsborough Stadium disaster, 15 April 1989: inquiry by the Rt Hon Lord Justice Taylor : interim report, Her Majesty's Stationery Office, 1989.

[45] The Apache Software Foundation, Apache Hadoop, `http://hadoop.apache.org/`, 2013. [Online; accessed 2013-12-10].

[46] The Storm Project, Storm: Distributed and fault-tolerant realtime computation, `http://storm-project.net/`, 2013. [Online; accessed 2013-12-10].

[47] P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in: Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2001), volume 1, pp. 511–518.

[48] G. Welch, G. Bishop, An introduction to the Kalman filter, Technical report, TR-95041, Department of Computer Science, University of North Carolina (2004).

[49] L.A. Zadeh, Fuzzy logic, neural networks, and soft computing, Commun. ACM 37 (1994) 77–84.

[50] P. Zhang, T. Thomas, S. Emmanuel, M.S. Kankanhalli, Privacy preserving video surveillance using pedestrian tracking mechanism, in: Proceedings of the 2nd ACM Workshop on Multimedia in Forensics, Security and Intelligence, MiFor '10, ACM, New York, NY, USA, 2010, pp. 31–36.

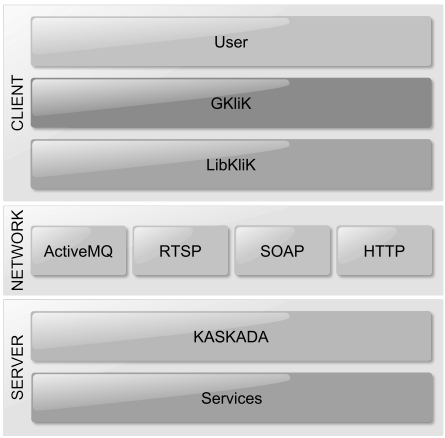

Figure 1: The layered model based on the client-server architecture providing the user with functionalities offered by the KASKADA platform

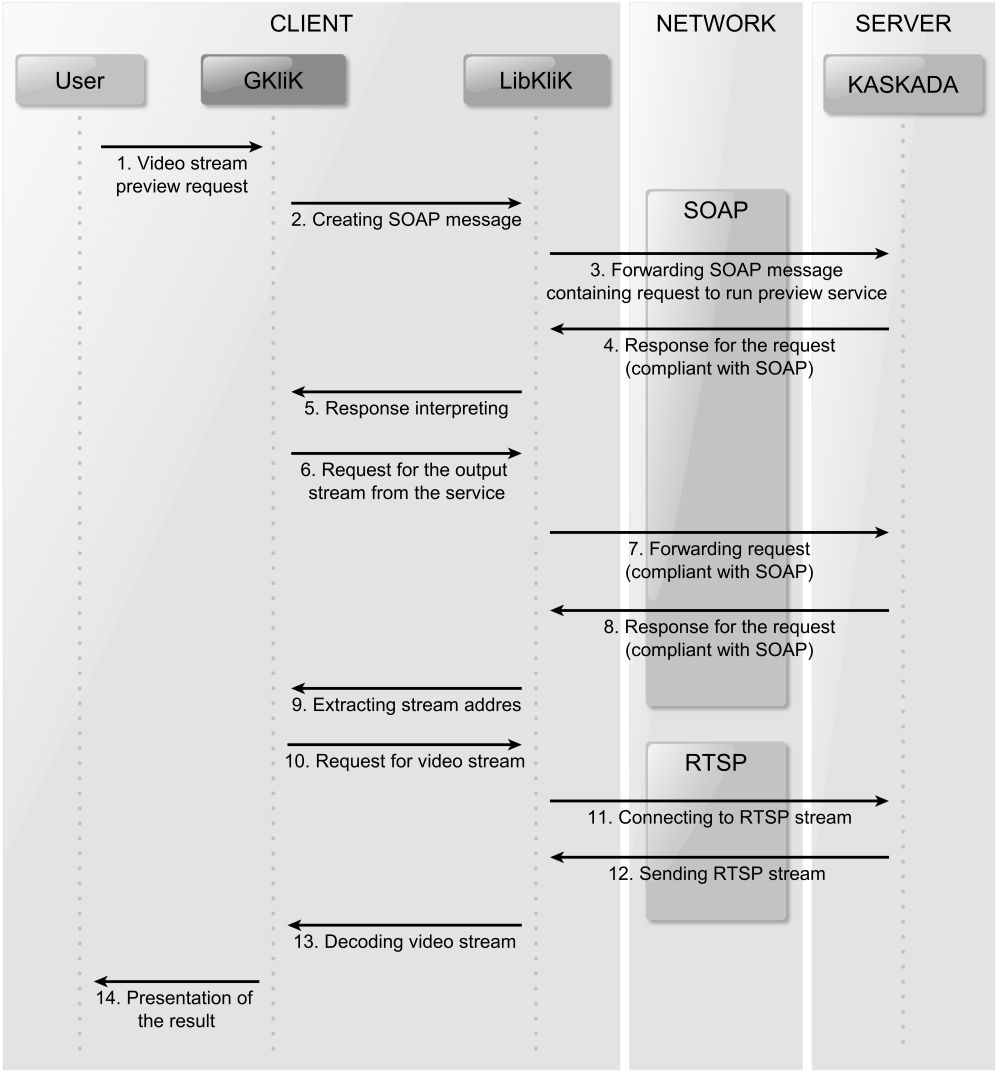

Figure 2: The example of handling a service request

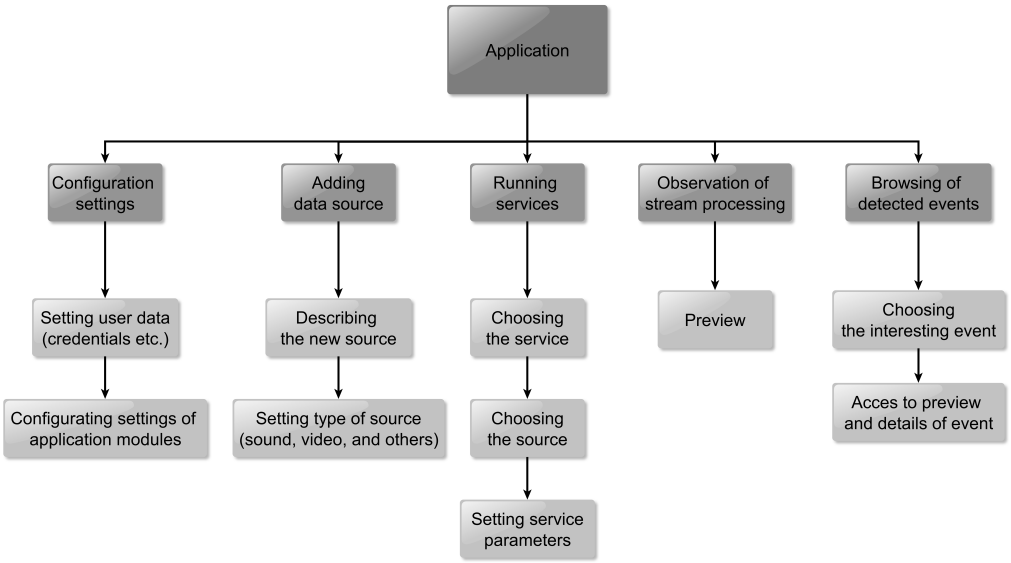

Figure 3: Main functionalities provided by the user application

Figure 4: Example of video analysis results: background subtraction, morphological processing, object detection and tracking, event detection

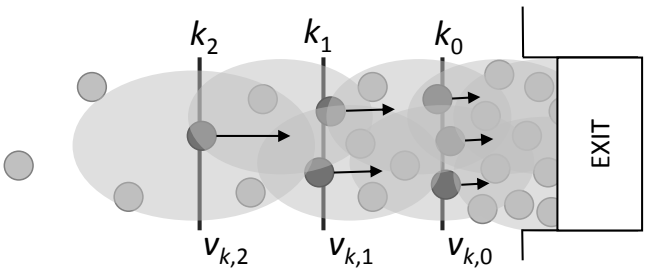

Figure 5: The concept of the crowding detector: k_0, k_1, k_2–control lines, $v_{k_0}, ...$–pedestrian flow speed

(a) Image recorded by camera (b) Motion map obtained by optical flow

Figure 6: Detection of crowding - original image (a) and processing result (b)

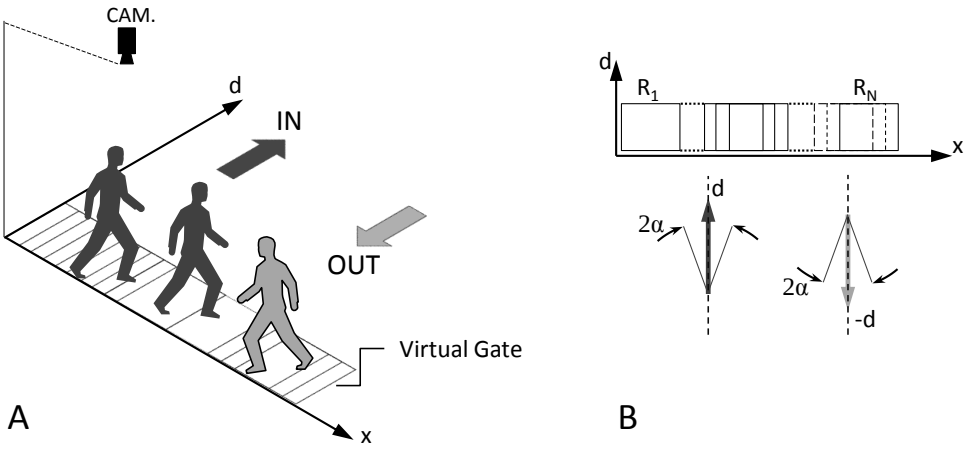

Figure 7: Virtual Gate: setup for counting people (A) and details illustrating its principle of working (B)

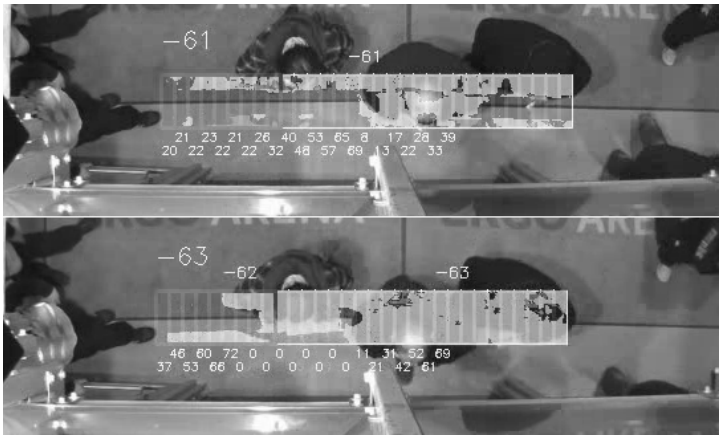

Figure 8: Result of counting people walking close each other

Figure 9: Results of face detector (left) and plate detector (right)

Figure 10: Real life example of the reversible anonymization: original frame (left), anonymized frame (center), deanonymized frame (right)

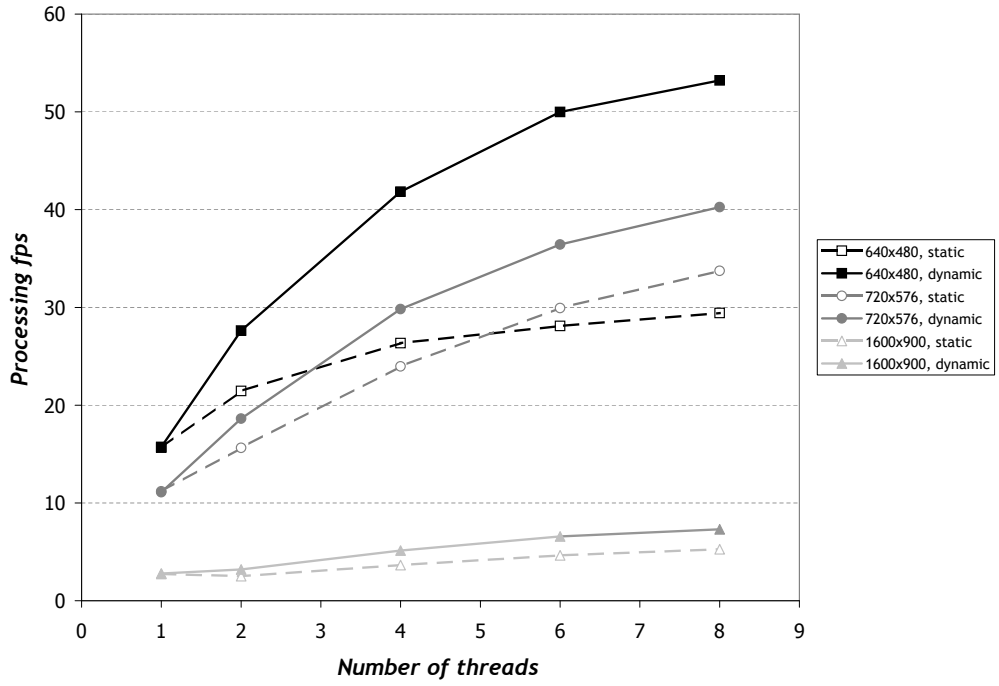

Figure 11: Results of object detection efficiency testing – frames processed per second vs the number of cores

a b c

Figure 12: Privacy enhancement with pixel relocation algorithm: a) faces, b) license plates, c) human silhouettes

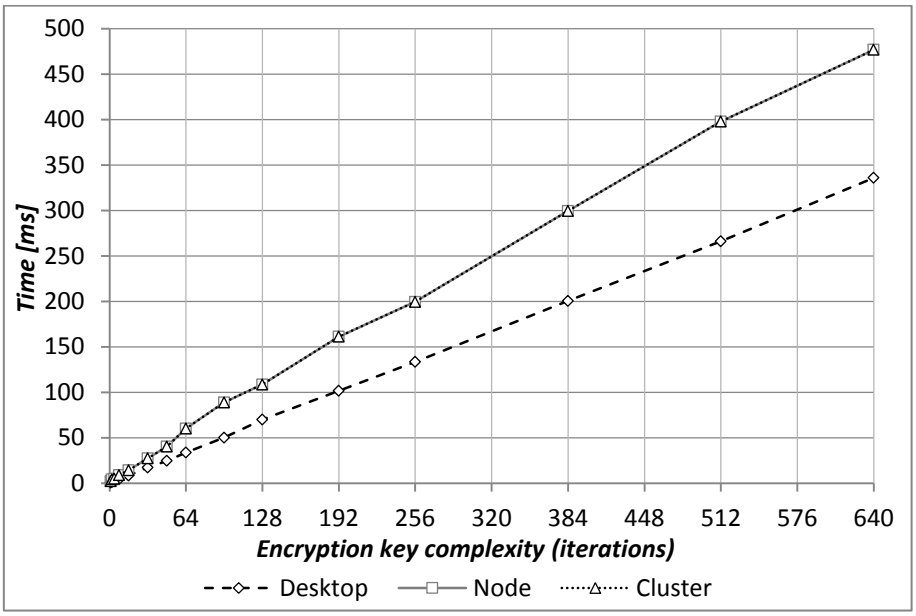

Figure 13: Time of processing according to encryption key complexity

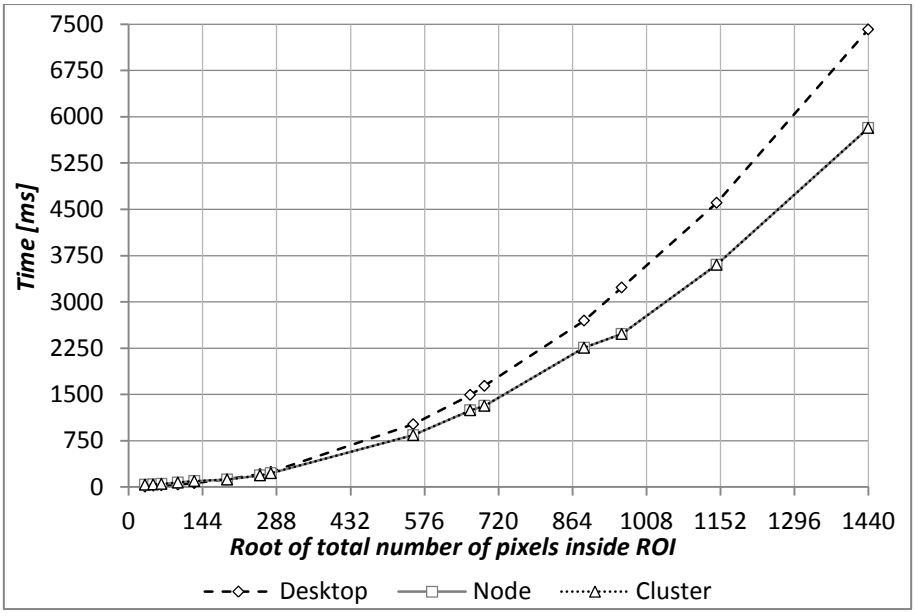

Figure 14: Time of processing according to object dimensions

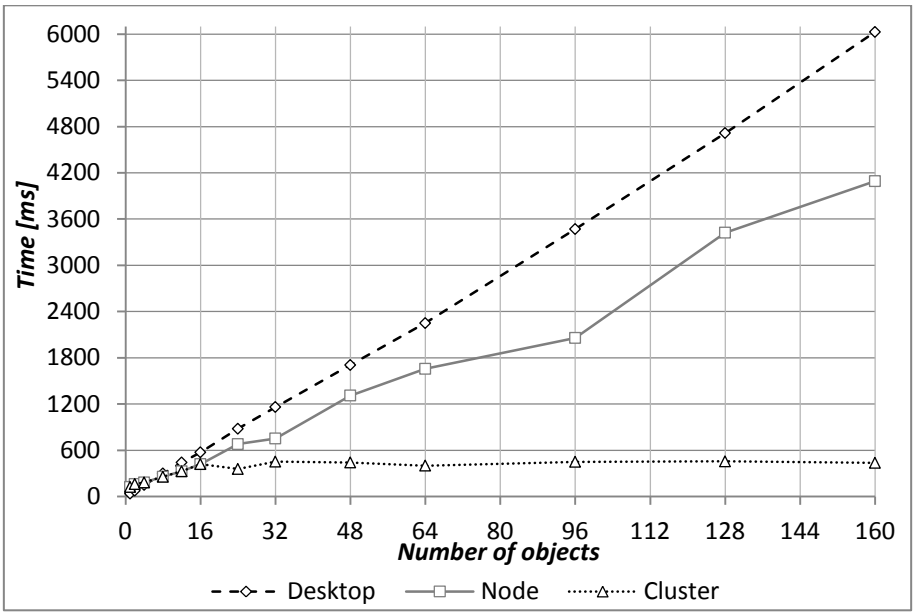

Figure 15: Time of processing according to the number of objects

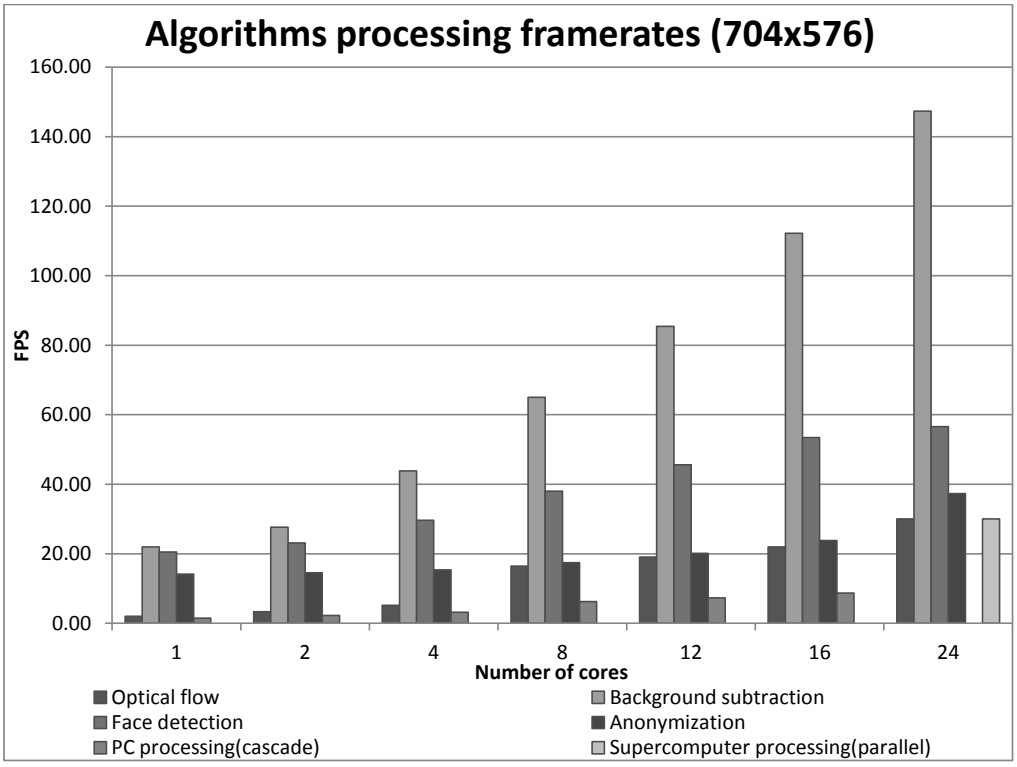

Figure 16: Algorithms processing framerates for PAL resolution data streams

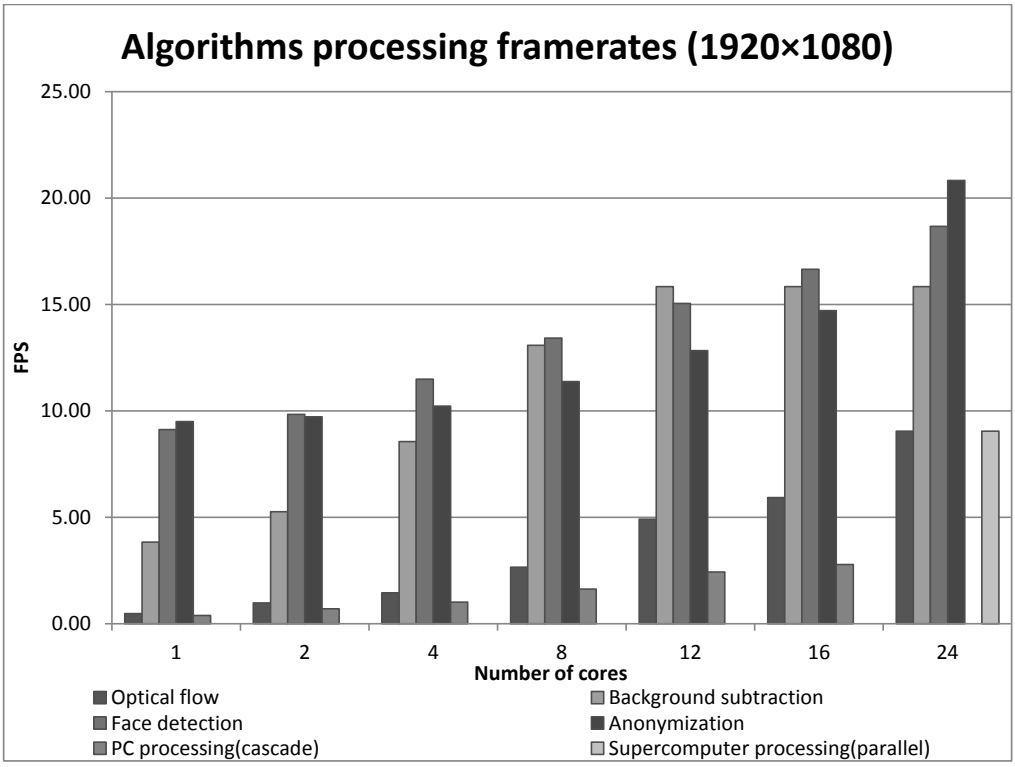

Figure 17: Algorithms processing framerates for HD resolution data streams