J. Sens. Actuator Netw. **2014**, *3*, 95-112; doi:10.3390/jsan3020095

OPEN ACCESS

Journal of Sensor and Actuator Networks

ISSN 2224-2708

www.mdpi.com/journal/jsan/

Article

Parallel Computational Intelligence-Based Multi-Camera Surveillance System

Sergio Orts-Escolano [1], Jose Garcia-Rodriguez [1,*], Vicente Morell [2], Miguel Cazorla [2], Jorge Azorin [1] and Juan Manuel Garcia-Chamizo [1]

[1] Computer Technology Department, University of Alicante, Po. Box 99, 03080 Alicante, Spain; E-Mails: sorts@dtic.ua.es (S.O.-E.); jazorin@dtic.ua.es (J.A.); juanma@dtic.ua.es (J.M.G.-C.)

[2] Artificial Intelligence Department, University of Alicante, Po. Box 99, 03080 Alicante, Spain; E-Mails: vmorell@dccia.ua.es (V.M.); miguel@dccia.ua.es (M.C.)

* Author to whom correspondence should be addressed; E-Mail: jgarcia@dtic.ua.es; Tel.: +34-965-903-400 (ext. 2616); Fax: +34-965-909-643.

Received: 16 December 2013; in revised form: 19 March 2014 / Accepted: 1 April 2014 / Published: 11 April 2014

Abstract: In this work, we present a multi-camera surveillance system based on the use of self-organizing neural networks to represent events on video. The system processes several tasks in parallel using GPUs (graphic processor units). It addresses multiple vision tasks at various levels, such as segmentation, representation or characterization, analysis and monitoring of the movement. These features allow the construction of a robust representation of the environment and interpret the behavior of mobile agents in the scene. It is also necessary to integrate the vision module into a global system that operates in a complex environment by receiving images from multiple acquisition devices at video frequency. Offering relevant information to higher level systems, monitoring and making decisions in real time, it must accomplish a set of requirements, such as: time constraints, high availability, robustness, high processing speed and re-configurability. We have built a system able to represent and analyze the motion in video acquired by a multi-camera network and to process multi-source data in parallel on a multi-GPU architecture.

Keywords: growing neural gas; camera networks; visual surveillance; GPU; CUDA; multi-core

1. Introduction

The development of the visual surveillance process in dynamic scenes often includes steps for modeling the environment, motion detection, classification of moving objects, tracking and the recognition of actions. Most of the work is focused on applications related to tracking people or vehicles that have a large number of potential applications, such as: controlling access to special areas, the identification of people, traffic analysis, anomaly detection, security management or interactive monitoring using multiple cameras [1,2].

The recognition of actions has been extensively investigated [3,4]. The analysis of the trajectory is also one of the basic problems in understanding the actions [5]. Relevant works on tracking objects can also be found in [6,7], among others.

Moreover, the majority of visual surveillance systems for scene analysis and surveillance depend on the use of knowledge about the scenes where the objects move in a predefined manner [8,9].

In recent years, work in the analysis of behaviors has been successful, because of the use of effective and robust techniques for detecting and tracking objects and people. This fact has allowed focused interest in higher levels of scene understanding. Moreover, thanks to the proliferation of low-cost vision sensors, embedded processors and the efficiency of wireless networks, a large amount of research has focused on the use of multiple sources of information for the analysis of behavior.

In particular, multi-camera networks are used for interpreting the dynamics of objects moving in wide areas or for observing objects from different viewpoints to achieve a 3D interpretation. Multiple viewpoints help in dealing with ambiguities and occlusions and can lead to a more reliable analysis of the scene.

Third generation surveillance systems usually refer to system conceived of to deal with a large number of cameras, a geographical spread of resources and many monitoring points and to mirror the hierarchical and distributed nature of the human process of surveillance [2].

With the deployment of camera networks in end-user environments, such as private homes or public places, awareness of privacy, confidentiality and general security issues is rising. Emphasis should be given to the special requirements of camera networks systems, including privacy and continuous real-time operation. To guarantee data authenticity and to protect sensitive and private information, a wide range of mechanisms and protocols should be included in the design surveillance systems based on camera networks.

The intrinsically parallel behavior of artificial neural networks, which were used to represent video events in our previous work [10], permits their implementation on GPUs (graphic processor units), allowing the processing of multiple camera images simultaneously. In the last few years, the high computing performance of GPUs and the low cost of these devices attracted a big number of researchers. Implementations related to neural networks [11–13] or computer vision and image processing [14–21] are of particular interest in this work.

Moreover, GPUs have played a role in video and image processing for a long time. In the beginning, they were used to display the processed results. Quickly, application developers picked up GPU computing, and GPUs are becoming the main processing devices in today’s video and image processing applications. The ever-increasing amount of video and image data demands ever-increasing computational power, while offering, at the same time, more potential for parallel computation. The

GPU, with its many-core architecture, is the perfect match to these challenges and delivers the computational performance necessary to drive the image and video-processing applications [22].

To accomplish the acceleration of the neural network learning algorithm used in our previous single-camera surveillance system work [23] and its application to the multi-camera video surveillance system with strong temporal constrains, a redesign of the sequential algorithm has been carried out, executed on the CPU to exploit the parallelism offered by the GPU. Moreover, multi-core CPU architecture and multiple GPU devices have been combined to manage several streams in parallel and also to accelerate each stream using different GPUs. Current GPUs have a large number of processors that can be used for general purpose computing. The GPU is specifically well suited to solving computationally intensive problems that can be expressed as data parallel computations [24]. However, implementation on a GPU requires the redesign of the algorithms, focused and adapted to its architecture. In addition, the programming of these devices has different restrictions, such as: the need for high computation in each processor to hide latencies produced by constrained memory access, management and synchronization of different threads running simultaneously, the proper use of the hierarchy of memories and other considerations. Researchers have already successfully applied GPU computing to problems that were traditionally addressed by the CPU [22,25].

The GPU implementation used in this work is based on NVIDIA’s CUDA (Computed Unified Device Architecture) architecture [26], which is supported by most current NVIDIA graphics chips. Supercomputers that currently lead the world ranking combined the use of a large number of CPUs with a high number of GPUs.

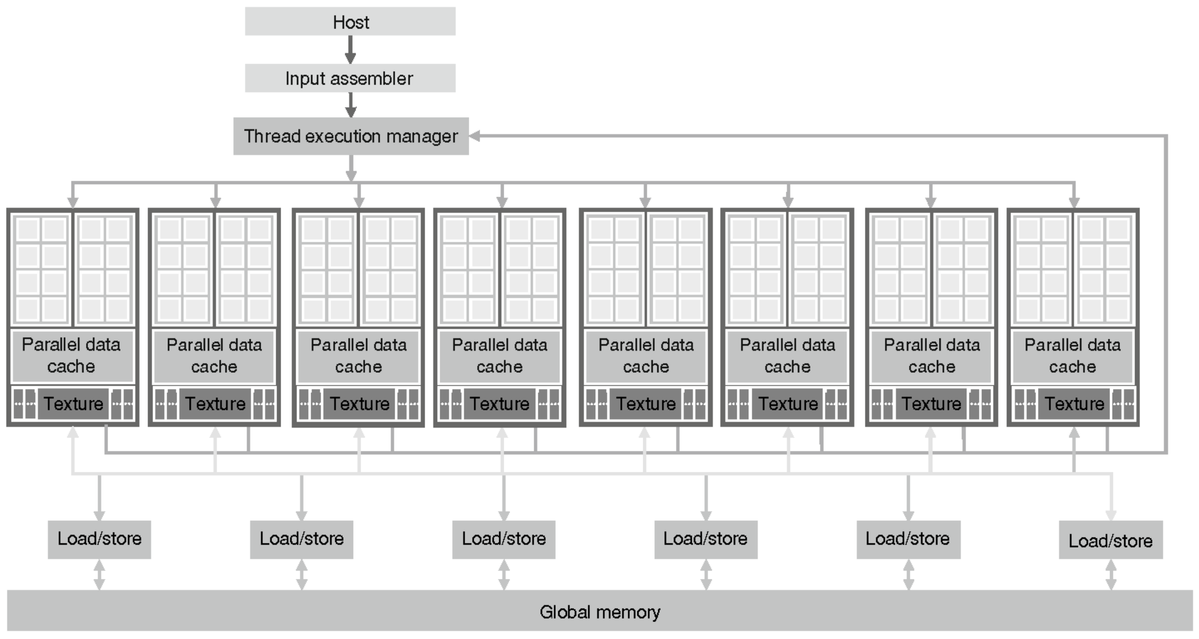
The remainder of the paper is organized as follows: Section 2 provides a description of the multi-core general purpose graphics processor unit (GPGPU) architecture. Section 3 presents the growing neural gas (GNG) algorithm and its modification to manage image sequences. In Section 4, we present the GPU parallel implementation of the GNG algorithm to accelerate the representation and tracking of objects in image sequences with a single camera. Section 5 shows the extension of this parallel implementation with multi-GPUs of a multi-camera visual surveillance system, followed by our major conclusions and further work.

2. Multicore GPGPU Architecture

A CUDA-compatible GPU is organized as a set of multiprocessors, as shown in Figure 1 [27]. These multiprocessors, called streaming multiprocessors (SMs), are highly parallel at the thread level. However, the number of multiprocessors varies depending on the generation of the GPU. Each SM consists of a series of streaming processors (SPs) that share the control logic and cache memory. Each of these SPs are launched in parallel with a huge amount of threads. For instance, GT200 graphics chips, with 240 SPs, are capable of performing a computing power of one teraflop, launching 1,024 threads per SM, with a total of 30,000 threads. The current GPUs have up to 12 GB of DRAM (Dynamic Random Access Memory), referenced in Figure 1 as global memory. The global memory is used and shared by all the multiprocessors, but it has a high latency.

The CUDA architecture reflects an SIMT model: single instruction, multiple threads. These threads are executed simultaneously working on a lot of data in parallel. Each of them runs a copy of the kernel (a piece of code that is executed) on the GPU and uses local indexes to be identified. Threads are grouped into blocks to be executed. Each of these blocks is allocated to a single multiprocessor, enabling the execution of several blocks within a multiprocessor. The number of blocks that are executed depends on the available resources in the multiprocessor, scheduled by a system of priority queues.

Figure 1. CUDA compatible GPU (graphic processor unit) architecture.



Within each of these blocks, the threads are grouped into sets of 32 units to carry out fully parallel execution on processors. Each set of 32 threads is called a warp. In the architecture, there are certain restrictions on the maximum number of blocks, warps and threads on each multiprocessor, but it varies depending on the generation and model of the graphics cards. In addition, these parameters are set for each execution of a kernel to get the maximum occupancy of hardware resources and to obtain the best performance. The Experiments section shows how to fit these parameters to execute our GPU implementation.

CUDA architecture also has a memory hierarchy. Different types of memory are found: constant, texture, global, shared and local registries. The shared memory is useful for implementing caches. Texture and constant memory are used to reduce computational cost, avoiding global memory access, which has high latencies.

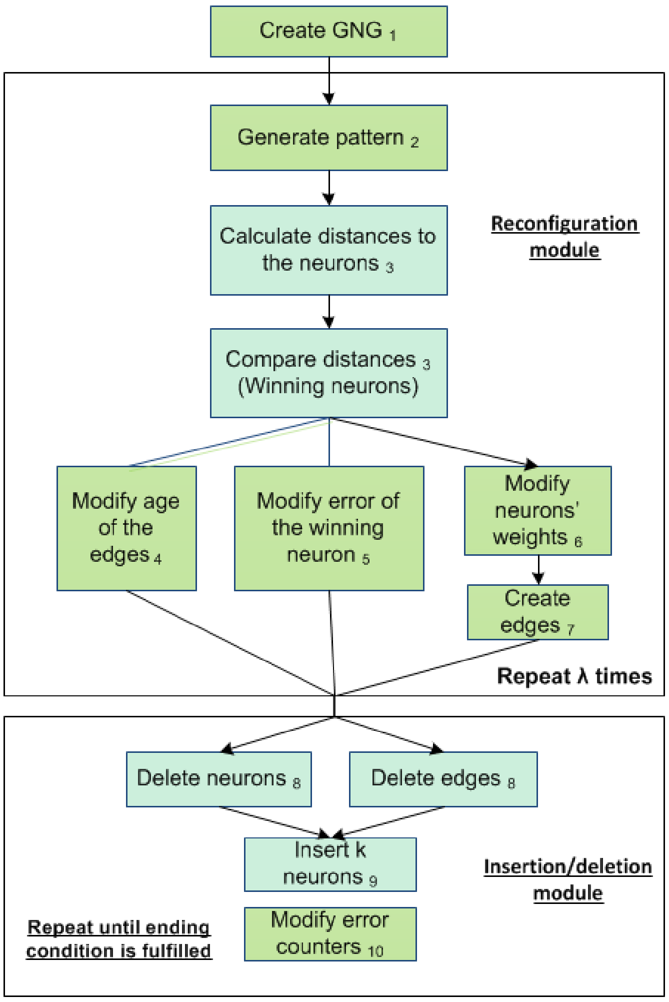
In recent years, a large number of applications have used GPUs to speed up the processing of neural network algorithms [11,15,28–30] applied to various computer vision problems, such as representation and tracking of objects in scenes [10], face representation and tracking [12] or pose estimation [17].

3. Growing Neural Gas and GNG-Seq

From the neural gas model [31] and growing cell structures [32], Fritzke developed the growing neural gas model [33], with no predefined topology of a union between neurons, to which, from an initial number of neurons, new ones are added (Figure 2).

A new version of the GNG model, called GNG-Seq (sequence), has been created to manage image sequences under a time constraint. Taking advantage of the GNG representation, obtained in previous frames, and considering that at video frequency, the slight motion of mobile agents can be modeled, it is only necessary to relocate a neural network structure without the necessity of adding or deleting neurons.

Figure 2. Growing neural gas (GNG) learning algorithm showing the parallel stages.


Create GNG 1
Generate pattern 2
Calculate distances to the neurons 3
Reconfiguration module
Compare distances 3 (Winning neurons)
Modify age of the edges 4
Modify error of the winning neuron 5
Modify neurons' weights 6
Create edges 7
Repeat λ times
Delete neurons 8
Delete edges 8
Insert k neurons 9
Insertion/deletion module
Repeat until ending condition is fulfilled
Modify error counters 10

3.1. GNG Algorithm

GNG is an unsupervised incremental clustering algorithm that, given some input distribution in R^d, incrementally creates a graph, or network of nodes, where each node in the graph has a position in R^d. The model can be used for vector quantization by finding the code vectors in clusters. These code vectors are represented by the reference vectors (the position) of the nodes. They can also be used for finding topological structures that closely reflect the structure of the input distribution. GNG learning is a dynamic algorithm in the sense that if the input distribution slightly changes over time, it is able to adapt, moving the nodes to the new input space $R^{d'}$.

Starting with two nodes, the algorithm constructs a graph in which nodes are considered neighbors if they are connected by an edge. The neighbor information is maintained throughout the execution by a variant of competitive Hebbian learning (CHL).

The graph generated by CHL creates an “induced Delaunay triangulation” that is a sub-graph of the Delaunay triangulation corresponding to the set of nodes. The induced Delaunay triangulation optimally preserves the topology in a very general sense [34]. CHL is an essential component of the GNG algorithm, since it is used to manage the local adaptation of nodes and the insertion of new nodes.

The network is specified as:

(1) A set, N, of nodes (neurons). Each neuron $c \in N$ has its associated reference vector $w_c \in R^d$. The reference vectors can be regarded as positions in the input space of their corresponding neurons.

(2) A set of edges (connections) between pairs of neurons. These connections are not weighted, and their purpose is to define the topological structure. An edge aging scheme is used to remove connections that are invalid, due to the motion of the neuron during the adaptation process.

GNG uses parameters that are constant in time. Further, it is not necessary to decide the number of nodes used *a priori*, since nodes are added incrementally during the execution. Insertion of new nodes stops when a user-defined performance criterion is fulfilled or alternatively when a maximum network size has been reached.

The adaptation of the network to the input space vectors is produced in Step 6. The insertion of connections (Step 4) between the winning neuron and the second closest to the input signal provides the topological relationship between the neurons (Figure 2).

The edge removal step (Step 8) eliminates the edges that are no longer part of that topology. This is done by removing the connections between neurons that are no longer near or that have other neurons that are closer, so that the age of these connections exceeds a threshold.

The accumulation of the error (Step 5) can identify those areas of the input space of vectors where it is necessary to increase the number of neurons to improve the mapping.

3.2. GNG-Seq

To analyze the movement, for each image in a sequence, objects are tracked following the representation obtained using the neural network structure. Therefore, the position or neuron reference vectors are used as stable markers in the tracking process. It is necessary to obtain a representation or graph that defines the position and shape of the object at each input frame in the sequence.

One of the most advantageous features of the GNG is that it is not required to restart the learning of the network for each input frame in the sequence. A previous neural network structure, obtained from previous frames, is used as a starting point for new frames representation, provided that the sampling rate is sufficiently high. In this way, a prediction based on historical images and a small re-adjustment of the network provides a new representation in a very short time (total learning time/N), where total learning time is the complete learning algorithm that linearly depends on the number of input patterns, λ, and the number of neurons, N. This provides a very high processing speed. This GNG-based architecture for the representation and processing of image sequences has been called GNG for sequences or GNG-Seq.

The process of tracking an object in each image is based on the following steps:

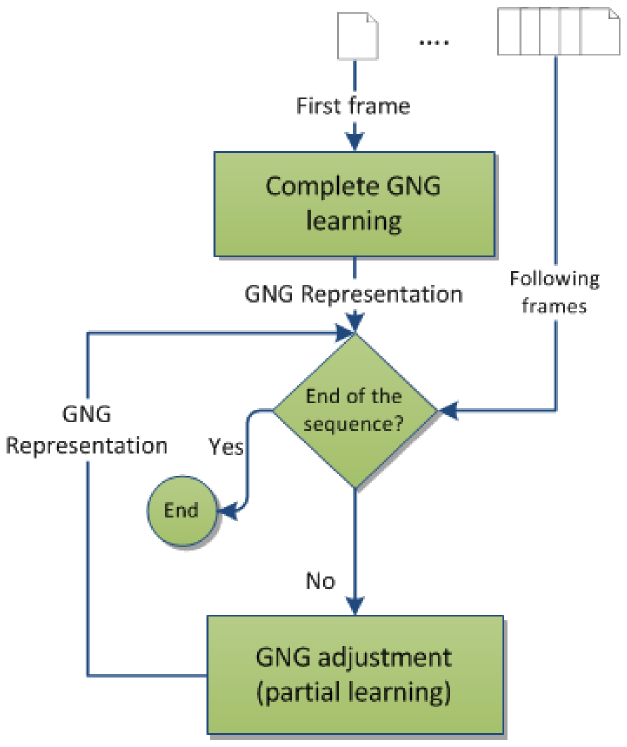
(1) Calculation of the transformation function, ψ, to segment the object from the background based on information from previous frames stored in the neural network structure.

(2) Prediction of the new neuron reference vectors.

(3) Re-adjustment of neuron reference vectors.

Figure 3 outlines the process to track objects, which differentiates the processing of the first frame, since no previous data is available and is needed to learn the complete network. In the second level, it estimates and updates the positions of neurons (reference vectors) depending on the available information on previous frames in the sequence stored in the neural network structure. In this way, the objects or agents are segmented in the next frame, estimating the new position and readjusting the map based on the information available from previous maps.

Figure 3. GNG-Seq (sequence) flowchart.



The complete GNG algorithm presented in Figure 2 is used to obtain the representation for the first frame of the image sequence, performing the full learning process. However, for the next frames, the final positions (reference vectors) of the neurons obtained from the previous frame are used as the starting point, performing only the reconfiguration module of the general algorithm, with no insertion or deletion of neurons, but moving neurons and deleting edges where necessary.

Further details of the image processing aspects of the surveillance system can be found in our previous work [10]. Furthermore, in [35], a comparison with other tracking and motion estimation methods can be found that demonstrates the accuracy of the system with a single camera.

4. GPU Implementation of GNG

The GNG learning algorithm has a high computational cost. For that reason, it is proposed to accelerate it using GPUs and to take advantage of the many-core architecture provided by these devices, as well as their parallelism at the instruction level. GPUs are specialized hardware for computationally intensive high-level parallelism that use a higher number of transistors to process data and less for flow control or management of the cache, unlike in CPUs. We have used the architecture and set of programming tools (language, compiler, development environment, debugger, libraries, *etc*.) provided by NVIDIA to exploit the parallelism of its hardware.

To accelerate the GNG algorithm on GPUs using CUDA, it was necessary to redesign it so that it fit within the GPU architecture. Many of the operations performed in the GNG algorithm are parallelizable, because they act on all the neurons of the network simultaneously. That is possible, because there is no direct dependence between neurons at the operational level. However, there exists a dependence on the adjustment of the network, which affects the end of each iteration and forces one to synchronize various parallel execution operations. Figure 2 shows GNG algorithm steps that have been accelerated on the GPU using kernels.

4.1. Euclidean Distance Calculation

The first stage of the algorithm that has been accelerated is the calculation of Euclidean distances performed each iteration. This stage calculates the Euclidean distance between a random pattern and each of the neurons. This task may take place in parallel by running the calculation on each neuron's distance on as many threads as the neurons the network contains. It is possible to calculate more than one distance per thread, but this is only efficient for large vectors, where the number of blocks that are executed on the GPU is also very high (>100,000).

4.2. Parallel Reduction

The second task parallelized is the search of the winning neuron: the one with lower Euclidean distance to the pattern generated and the second closest. For the winning search, we used a parallel reduction technique described in [36]. This technique accelerates operations, such as the search for the minimum value in parallel on large data sets. For our work, we modified the original algorithm that we have called *2 min parallel reduction*, so that, with a single reduction, not only is the minimum obtained, but the two smallest values of the entire data set. Parallel reduction is described as a binary tree, where: $\log_2(n)$ steps operated in parallel in sets of two elements, by applying an operation on these elements in parallel; at the end of the $\log_2(n)$ steps, we obtained the final result of the operation on a set of N elements.

4.3. Other Optimizations

To speed-up the remaining steps, we have followed the same strategy used during the first phase. Each thread is responsible for performing an operation on a neuron: check the edge connection age and, in case it exceeded a certain threshold, delete it, update the local error of the neuron or adjust the neuron weights. In the stage of finding the neuron with maximum error, the same strategy used to find the winning neuron has also been followed, but in this case, the reduction is searching only for the neuron with the highest error.

Regardless of the parallelism of the algorithm, we have followed some good practices for the CUDA architecture to get more performance. First, we used constant memory to store the neural network parameters, ε_1, ε_2, α, β, α_{max}. By storing these parameters in this memory, the access is faster than working with values stored in the global memory.

Our GNG implementation on the GPU architecture is also limited by the memory bandwidth available. In the Experiments section, a specification report for each CUDA capable device used its memory bandwidth will be shown. Although this bandwidth is only attainable under highly idealized memory access patterns, it does provide us with an upper limit of memory performance. Moreover, some memory access patterns, like moving data from the global memory into shared memories and registers, provide better coalesced access. To get the highest advantage of the memory bandwidth, the shared memory within each multiprocessor has been used. Therefore, this memory acts as a cache to avoid frequent access to the global memory and allows threads to achieve coalesced reads when accessing neurons' data.

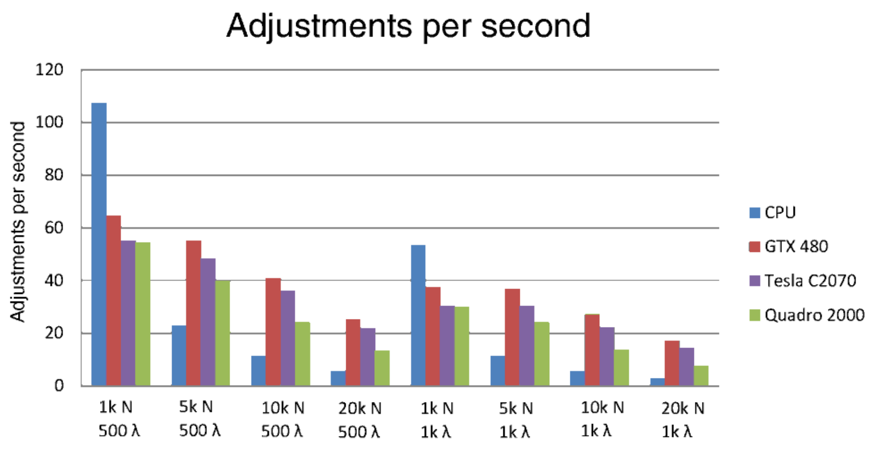
For instance, a GNG network composed of 20,000 neurons and auxiliary structures requires only 17 megabytes. Therefore, the GPU implementation, in terms of size, does not present problems, because currently, GPU devices have enough memory to store it.

Memory transfers between the CPU and GPU are the main bottlenecks to obtain a speed-up. These transfers have been avoided as much as possible. Initial versions of the algorithm failed to obtain performance over the CPU version, because the complete neural network was copied from the GPU memory to the CPU memory and *vice versa* for each input pattern generated. This penalty, introduced due to the bottleneck of the transfer through the PCI (Peripheral Component Interconnect)-Express bus, was so high that we did not initially improve the CPU version. After careful consideration about the flow of execution, we decided to move the inner loop of the pattern generation to the GPU, although some tasks are not parallelizable and had to be run on a single GPU thread.

4.4. Rate of Adjustments per Second

We performed several experiments in which it is shown how the accelerated GNG version performs a faster complete learning process than the CPU version. Moreover, it allows performing more adjustments per second. For instance, after learning a network of 20,000 neurons, the GPU implementation performs 17 adjustments per second, while the CPU version only achieves 2.8. This means that the GPU implementation samples more data during the learning process and, therefore, obtains a better topological representation than the CPU implementation. This faster learning process allows the use of this implementation under time constraints. Figure 4 shows the adjustment rate per second performed by different GPU devices and the CPU. It is also shown how increasing the number of neurons in the CPU cannot handle a high rate of adjustments per second.

Figure 4. The rate of adjustments per second performed by different GPU devices and the CPU with different numbers of neurons and input patterns (k indicates thousands).



5. Multisource Information Processing with GPU

We have considered the following domain-specific constraints regarding a high demand (25 frames per second) video surveillance application in our study. In order to achieve real-time processing, the number of frames processed per second (fps) was determined depending on the number of cameras. The maximum number of cameras that can be supported in the multi-GPU architecture is obtained by calculating the number of video frames that can be processed simultaneously with this constraint.

5.1. Tracking Multiple Agents in Visual Surveillance Systems

The technique used for tracking multiple objects is based on the use of the GNG-Seq, described in Section 3.2 and previous work [10], which with its fast algorithm, segments and tracks the different objects present in the image. Once the objects in the image are segmented, it is possible to identify groups of neurons that are mapping each of them and follow them separately. These groups will be identified and labeled to use them as a reference keeping the correspondences between frames.

The system has several advantages compared to other tracking systems:

- The graph obtained with the neural network structure permits the representation of local and global movement.
- The information stored in the structure of the neural network through the sequence permits the representation of motion and the analysis of entity behavior based on the trajectory followed by the neural network nodes.
- The feature correspondence problem is solved using the structure of the neural network.
- Real-time processing is supported, since the accelerated neural network allows obtaining fast representations, depending on the available time.

However, an important drawback should be considered:

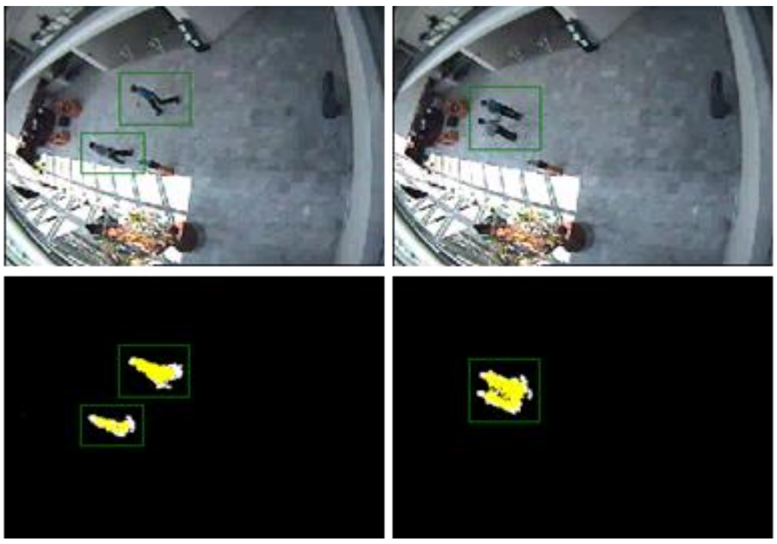
- The quality of the representation is highly dependent on the robustness of the segmentation process.

5.2. Privacy and Security: Real-Time Constraint

Real-time processing is of great importance, particularly in the security domain. A high number of cameras, combined with an increase in security risks in crowded public places, such as airports, underground stations or terminals and town squares, requires the automation of the surveillance process in real time, because it is no longer possible to carry out this process manually. Real-time processing is also required in other video applications, such as automated video content analysis or robotics.

Figure 5 presents the representation of people appearing in the image with our system. The system is able to maintain the privacy of the persons under observation by using the graph representation provided by the GNG instead of the original video images.

Figure 5. GNG graph representation.



The importance of GPUs has recently been recognized for different applications, such as video and image processing algorithms; in particular, real-time video surveillance applications [37,38], where it is possible to manage information from different cameras [39], thanks to the parallel processing capabilities of GPUs.

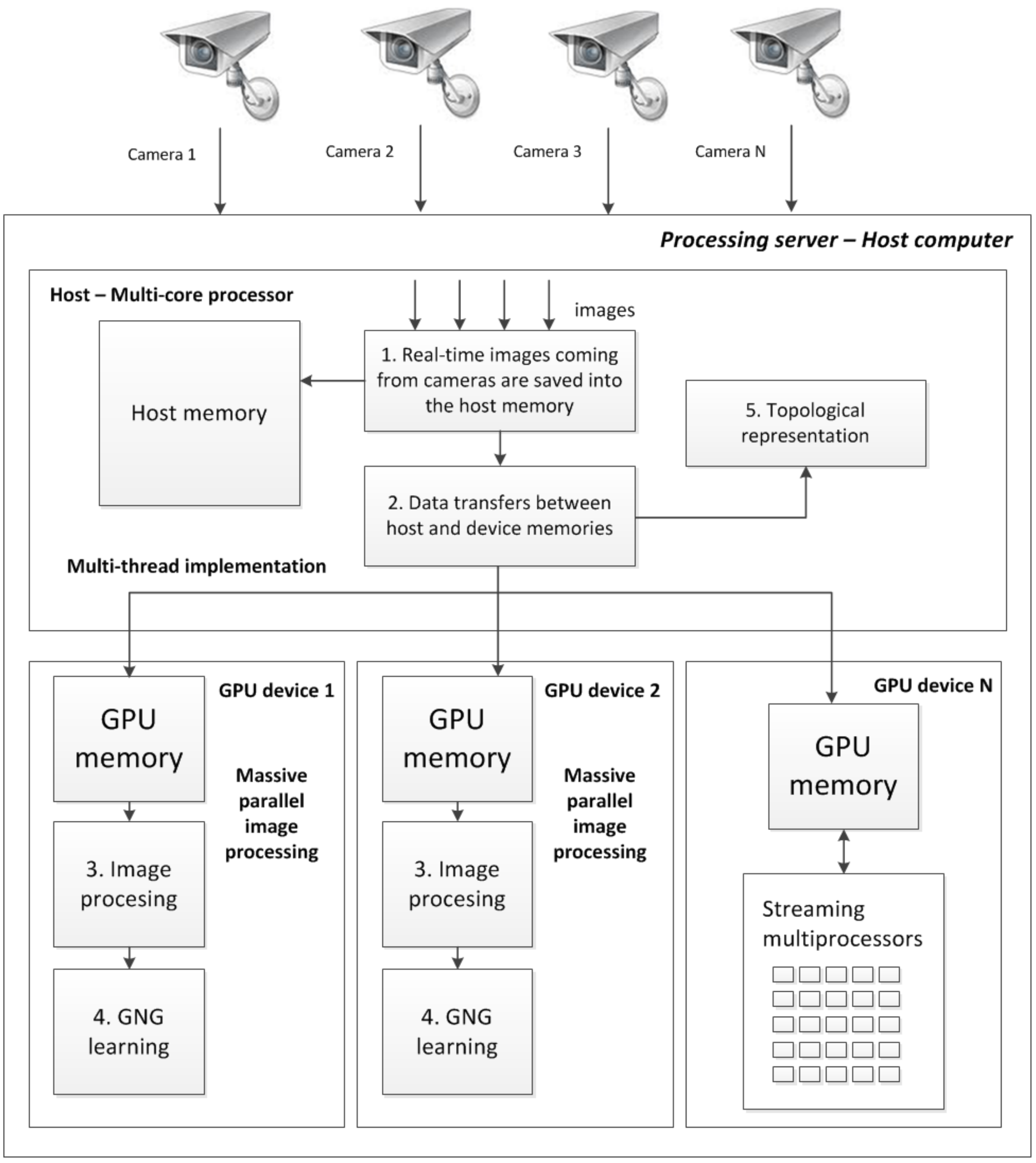
The processing of information from different cameras in the proposed architecture allows one to solve problems like occlusions or to interpret simultaneous actions in different areas under surveillance.

5.3. Multi-Core CPU and Multi-GPU Implementation

The processing of information from different cameras on a general purpose graphics processing unit (GPGPU) architecture allows fast access to the data obtained from different sources. These data are allocated in centralized shared memories and permit solving problems, like occlusions or interpreting concurrent actions in different areas under surveillance. This is possible, since the compact representation obtained with the GNG permits the storage of a large amount of information.

We first developed a single camera GPU version, where the algorithm was parallelized and accelerated via CUDA technology. After accelerating image processing on a single node using a GPU, we implemented a higher level of abstraction. This new implementation includes the use of a multicore processor and various GPUs connected to the same host system (Figure 6). Through an efficient management of multi-core CPU threads and assigning different streams on different GPUs, we can perform the parallel processing of different streams faster than using a traditional processing system composed of a single CPU. This proposed system is also fully scalable, allowing the user to add more CPU cores and GPU cards to further improve the number of streams that can be managed in parallel. The POSIX (Portable Operating System Interface) Pthreads standard has been used to manage several CPU threads in parallel, performing different processing tasks in each GPU device.

Figure 6. Multi-GPU multi-camera workflow.


Camera 1
Camera 2
Camera 3
Camera N
Processing server – Host computer
Host – Multi-core processor
images
Host memory
1. Real-time images coming from cameras are saved into the host memory
5. Topological representation
2. Data transfers between host and device memories
Multi-thread implementation
GPU device 1
GPU memory
Massive parallel image processing
3. Image procesing
4. GNG learning
GPU device 2
GPU memory
Massive parallel image processing
3. Image procesing
4. GNG learning
GPU device N
GPU memory
Streaming multiprocessors

5.4. Experiments

In this section we present some results of experiments with the GNG used to learn features in 2D images with different numbers of cameras, neurons and input patterns based on our previous GNG visual surveillance system with a single camera [10]. Figure 7 shows the multi-target representation using our previous system for images from the database, CAVIAR (Context Aware Vision using Image-based Active Recognition) [40].

Figure 7. Example of a single-image multi-target visual surveillance system.

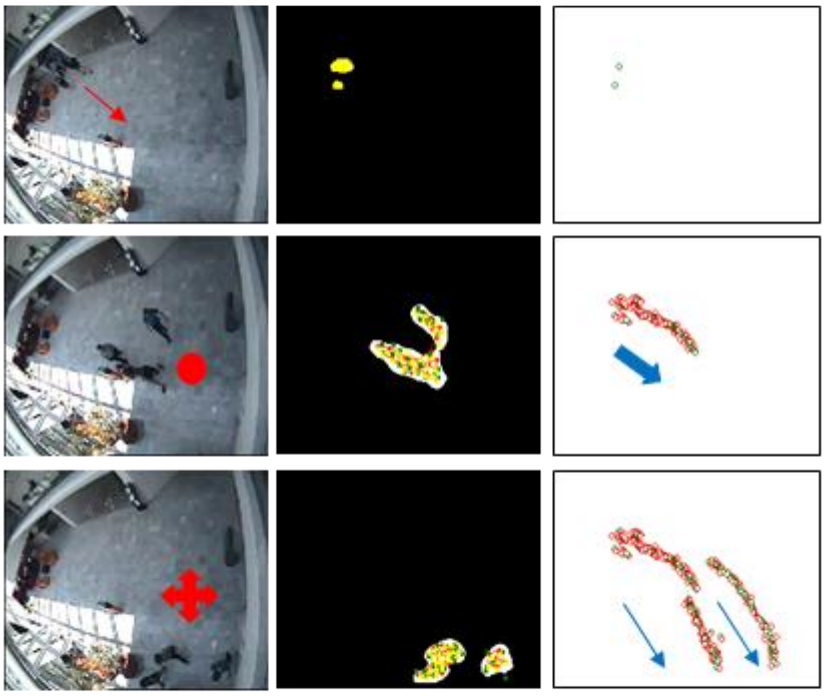


Figure 8 shows the results of experiments with different numbers of cameras to compare the time spent to characterize entities with GNG using 5,000 and 20,000 neurons, respectively, and training the maps with 500 and 1,000 input patterns per iteration. Experiments were developed on a computing node with a heterogeneous CPU-GPU architecture with multi-core CPUs and two GPU devices. Table 1 shows the specification of each GPU device installed in the proposed architecture and the multi-core processor installed.

Table 1. GPU device and multi-core processor specifications. SM, streaming multiprocessor.

Device	Capability	SMs	Cores	Memory	Bandwidth
GTX 480	2.0	15	480	1.5 GB	177.4 GB/s
Processor	#Cores	#Threads	Clock	Cache	Memory
Intel i3-540	2	4	3.06 GHz	4 MB	8 GB

It can be appreciated that the single-threaded CPU solution takes more and more time as the number of cameras grows. It can be seen how the CPU version time complexity grows much faster than the multi-core CPU and multi-GPU versions, and it is therefore an inadequate solution for large numbers of cameras. However, the proposed parallel version increases the number of cameras without degrading performance significantly. We also appreciated that, using a higher number of cameras, the speed-up obtained using the proposed architecture is about 2.5× faster compared with the single-threaded CPU version.

Figure 8. Execution times and speed-up for different GNG learning parameters and different numbers of cameras.

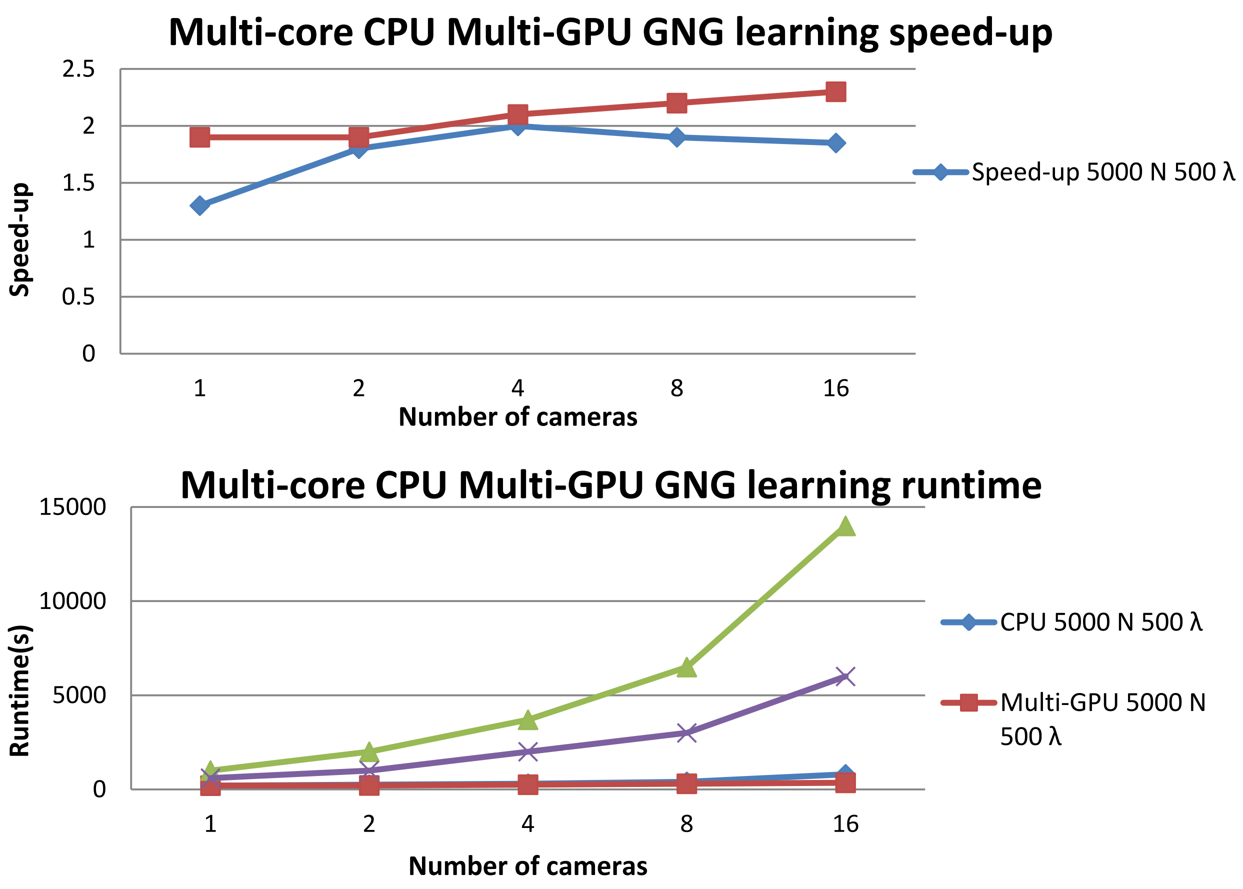
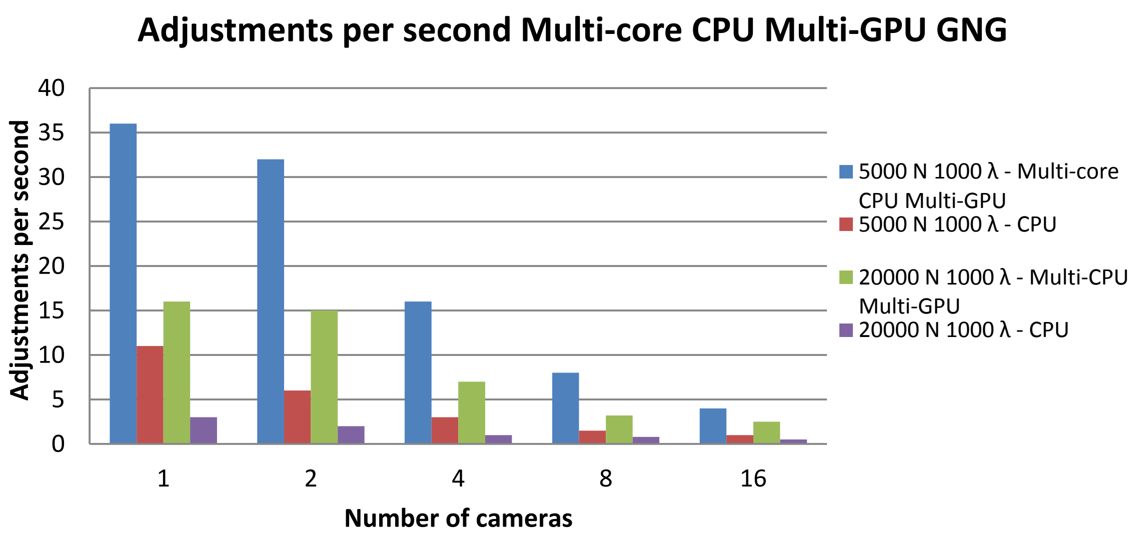


Figure 9. The rate of adjustments per second for different GNG learning parameters and different numbers of cameras using the multi-GPU-multi-CPU and single-threaded CPU versions.



In Figure 9, it is demonstrated that the proposed implementation is capable of processing in parallel up to 16 streams from different cameras and to perform 3 adjustments per second on each stream, compared with the single-threaded CPU version that only can perform adjustments on one stream. Moreover, it is shown that the CPU version obtains a considerably lower rate of adjustments compared with the proposed version, obtaining rates between 5 and 10 times higher.

6. Conclusions

In this work, a multi-core processor and a multi-GPU system based on the GNG neural network has been presented that is capable of representing mobile agents under time constraints supporting the parallel management of multiple information sources.

Our main contribution is the modification and acceleration of the GNG algorithm using multi-CPU and multi-GPU architectures in order to process several streams from multiple cameras.

Through the implementation of surveillance applications, the capabilities of the system for tracking and motion analysis have been demonstrated. The system automatically handles the mergers and splits among mobile agents that appear in the images and can detect and interpret the actions that are performed in video sequences. Moreover, the graph representation provided by GNG guarantees the privacy of the persons under observation.

As demonstrated in the experiments of the GPGPU implementation, the runtime of the sequential GNG algorithm grows as the number of neurons in the network increases. While in the parallel version, implemented on a GPU architecture, as we increase the number of neurons, we obtain a greater acceleration over the sequential version.

In experiments of multi-core CPU and multi-GPU GNG implementation, the ability to manage several streams from different cameras is demonstrated without degrading performance significantly compared with the CPU version. The results show that the parallel version can manage up to 16 cameras, processing a network composed of 5,000 neurons in each stream, performing three adjustments per image received, while the CPU version only can manage one camera with the same time constraint.

Acknowledgments

This work was partially funded by the Spanish Government DPI2013-40534-R grant and Valencian Government GV/2013/005 and University of Alicante UA GRE11-01 grants. Experiments were made possible with a generous donation of hardware from NVDIA.

Author Contributions

The work presented here was carried out in collaboration between all authors.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Hu, W.H.W.; Tan, T.T.T.; Wang, L.W.L.; Maybank, S.M.S. A Survey on Visual Surveillance of Object Motion and Behaviors. *IEEE Trans. Syst. Man Cybern. Part C.* **2004**, *34*, 334–352.
2. Velastin, S.A.; Remagnino, P. *Intelligent Distributed Video Surveillance Systems*; IET Digital Library: London, UK, 2006.
3. Collins, R.T.; Lipton, A.J.; Kanade, T. Introduction to the Special Section on Video Surveillance. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 745–746.
4. Howarth, R.J.; Buxton, H. Conceptual Descriptions from Monitoring and Watching Image Sequences. *Image Vis. Comput.* **2000**, *18*, 105–135.
5. Hu, W.; Xie, D.; Tan, T. A Hierarchical Self-Organizing Approach for Learning the Patterns of Motion Trajectories. *IEEE Trans. Neural Netw.* **2004**, *15*, 135–144.
6. Tian, Y.; Tan, T.N.; Sun, H.Z. A Novel Robust Algorithm for Real-Time Object Tracking. *Acta Autom. Sin.* **2002**, *28*, 851–853.
7. Wu, Y.; Liu, Q.; Huang, T.S. An Adaptive Self-Organizing Color Segmentation Algorithm with Application to Robust Real-Time Human Hand Localization. In Proceedings of 4th Asian Conference on Computer Vision, Taipei, Taiwan, 8–11 January 2000; pp. 1106–1111.
8. Howarth, R.J.; Buxton, H. Analogical Representation of Space and Time. *Image Vis. Comput.* **1992**, *10*, 467–478.
9. Brand, M.; Kettnaker, V. Discovery and Segmentation of Activities in Video. *IEEE Trans. Pattern Anal. Mach. Intell.* **2000**, *22*, 844–851.
10. Garcia-Rodriguez, J.; Garcia-Chamizo, J.M. Surveillance and Human-Computer Interaction Applications of Self-Growing Models. *Appl. Soft Comput.* **2011**, *11*, 4413–4443.
11. Nageswaran, J.M.; Dutt, N.; Krichmar, J.L.; Nicolau, A.; Veidenbaum, A. Efficient Simulation of Large-Scale Spiking Neural Networks Using CUDA Graphics Processors. In Proceedings of the 2009 International Joint Conference on Neural Networks, Atlanta, GA, USA, 14–19 June 2009; pp. 3201–3208.
12. Nasse, F.; Thurau, C.; Fink, G.A. Face Detection Using GPU-Based Convolutional Neural Networks. In Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns; Springer-Verlag: Berlin, Heidelberg, Germany, 2009; pp. 83–90.
13. Uetz, R. ; Behnke, S. Large-Scale Object Recognition with CUDA-Accelerated Hierarchical Neural Networks. In Proceedings of 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, Shanghai, China, 20–22 November 2009; Volume 1, pp. 536–541.
14. Che, S.; Boyer, M.; Meng, J.; Tarjan, D.; Sheaffer, J.W.; Skadron, K. A Performance Study of General-Purpose Applications on Graphics Processors Using CUDA. *J. Parallel Distrib. Comput.* **2008**, *68*, 1370–1380.
15. Jang, H.; Park, A.; Jung, K. Neural Network Implementation Using CUDA and OpenMP. In Proceedings of the 2008 Digital Image Computing: Techniques and Applications, Canberra, ACT, Australia, 1–3 December 2008; pp. 155–161.

16. Kim, J.; Hwangbo, M.; Kanade, T. Realtime Affine-Photometric KLT Feature Tracker on GPU in CUDA Framework. In Proceedings of IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), Kyoto, Japan, 27 September–4 October 2009; pp. 886–893.
17. Oh, S.; Jung, K. View-Point Insensitive Human Pose Recognition Using Neural Network and CUDA. *World Acad. Sci. Eng. Technol.* **2009**, *60*, 723–726.
18. Schwarz, M.; Stamminger, M. Fast GPU-Based Adaptive Tessellation with CUDA. *Comput. Gr. Forum* **2009**, *28*, 365–374.
19. Simek, V.; Asn, R.R. GPU Acceleration of 2D-DWT Image Compression in MATLAB with CUDA. In Proceedings of the 2008 2nd UKSIM European Symposium on Computer Modeling and Simulation, Liverpool, UK, 8–10 September 2008; pp. 274–277.
20. Stone, S.S.; Haldar, J.P.; Tsao, S.C.; Hwu, W.-M.W.; Sutton, B.P.; Liang, Z.-P. Accelerating Advanced MRI Reconstructions on GPUs. *J. Parallel Distrib. Comput.* **2008**, *68*, 1307–1318.
21. Hwu, W.W. *GPU Computing Gems Emerald Edition*, 1st ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2011.
22. Garcia-Rodriguez, J.; Angelopoulou, A.; García-Chamizo, J.M.; Psarrou, A.; Orts-Escolano, S.; Morell-Gimenez, V. Fast Autonomous Growing Neural Gas. In Proceedings of the 2011 International Joint Conference on Neural Networks (IJCNN), San Jose, CA, USA, 31 July–5 August 2011; pp. 725–732.
23. Nickolls, J.; Dally, W.J. The GPU Computing Era. *IEEE Micro* **2010**, *30*, 56–69.
24. Satish, N.; Harris, M.; Garland, M. Designing Efficient Sorting Algorithms for Manycore GPUs. In Proceedings of IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy, 23–29 May 2009; pp. 1–10.
25. CUDA Programming Guide, Version 5.0, 2013. Available online: http://docs.nvidia.com/cuda/cuda-c-programming-guide/ (accessed on 12 June 2013).
26. Kirk, D.B.; Hwu, W.W. *Programming Massively Parallel Processors: A Hands-on Approach*, 1st ed.; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2010.
27. Oh, K.-S.; Jung, K. GPU Implementation of Neural Networks. *Pattern Recognit.* **2004**, *37*, 1311–1314.
28. Juang, C.-F.; Chen, T.-C.; Cheng, W.-Y. Speedup of Implementing Fuzzy Neural Networks with High-Dimensional Inputs through Parallel Processing on Graphic Processing Units. *IEEE Tran. Fuzzy Syst.* **2011**, *19*, 717–728.
29. Garcia-Rodriguez, J.; Angelopoulou, A.; Morell, V.; Orts, S.; Psarrou, A.; Garcia-Chamizo, J.M. Fast Image Representation with GPU-Based Growing Neural Gas. In Proceedings of the 11th International Work-Conference on Artificial Neural Networks, Torremolinos-Málaga, Spain, 8–10 June 2011; pp. 58–65.
30. Igarashi, J.; Shouno, O.; Fukai, T.; Tsujino, H. Real-Time Simulation of a Spiking Neural Network Model of the Basal Ganglia Circuitry Using General Purpose Computing on Graphics Processing Units. *Neural Netw.* **2011**, *24*, 950–960.
31. Martinetz, T.M.; Berkovich, S.G.; Schulten, K.J. 'Neural-Gas' Network for Vector Quantization and Its Application to Time-Series Prediction. *IEEE Trans. Neural Netw.* **1993**, *4*, 558–569.
32. Fritzke, B. Growing Cell Structures—A Self-Organizing Network for Unsupervised and Supervised Learning. *Neural Netw.* **1993**, *7*, 1441–1460.

33. Fritzke, B. A Growing Neural Gas Network Learns Topologies. *Adv. Neural Inf. Process. Syst.* **1995**, *7*, 625–632.
34. Martinez, T. Competitive Hebbian Learning Rule Forms Perfectly Topology Preserving Maps. In *ICANN'93*; Springer: London, UK; pp. 427–434.
35. Garcia-Rodriguez, J.; Orts-Escolano, S.; Angelopoulou, A.; Psarrou, A.; Azorin-Lopez, J.; Garcia-Chamizo, J.M. Real Time Motion Estimation using a Neural Architecture Implemented on GPUs. *J. Real-Time Image Process.* **2014**, doi:10.1007/s11554-014-0417-y.
36. Harris, M. Optimizing Parallel Reduction in Cuda; In *NVIDIA Developer Technology*; NVIDIA Corporation: Santa Clara, CA, USA, 2007.
37. Schreiber, D.; Rauter, M. GPU-Based Non-Parametric Background Subtraction for a Practical Surveillance System. In Proceedings of 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops), Kyoto, Japan, 27 September–4 October 2009; pp. 870–877.
38. Wang, G.; Wong, T.-T.; Heng, P.-A. GPU-Friendly Warped Display for Scope-Maintained Video Surveillance. *Multimed. Syst.* **2006**, *12*, 169–178.
39. Temizel, A.; Halici, T.; Logoglu, B.; Temizel, T.T.; Omruuzun, F.; Karaman, E. Experiences on Image and Video Processing with CUDA and OpenCL. In *GPU Computing Gems Emerald Edition*; Hwu, W.W., Ed.; Morgan Kaufmann: Boston, MA, USA, 2011; pp. 547–567.
40. Fisher, R.B. PETS04 Surveillance Ground Truth Data Set. In Proceedings of the 6th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, Prague, Czech Republic, 10 May 2004; pp. 1–5.

© 2014 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/3.0/).