EFDA–JET–PR(10)01

S. Palazzo, A. Murari, G. Vagliasindi, P. Arena, D. Mazon, A. De Maack
and JET EFDA contributors

Image Processing with Cellular Nonlinear Networks Implemented on Field-Programmable Gate Arrays for Real Time Applications in Nuclear Fusion

Image Processing with Cellular Nonlinear Networks Implemented on Field-Programmable Gate Arrays for Real Time Applications in Nuclear Fusion

S. Palazzo[1], A. Murari[2], G. Vagliasindi[1], P. Arena[1], D. Mazon[3], A. De Maack[4]
and JET EFDA contributors*

JET-EFDA, Culham Science Centre, OX14 3DB, Abingdon, UK

[1]Dipartimento di Ingegneria Elettrica Elettronica e dei Sistemi-Universit‡ degli Studi di Catania, 95125 Catania, Italy
[2]Consorzio RFX-Associazione EURATOM ENEA per la Fusione, I-35127 Padova, Italy
[3]Association EURATOM-CEA, CEA Cadarache, 13108 Saint-Paul-lez-Durance, France
[4]École des Ponts ParisTech 6-8 avenue Blaise Pascal CitÈ Descartes - Champs-sur-Marne 77455 Marne-la-Vallée cedex 2
** See annex of F. Romanelli et al, “Overview of JET Results”, (Proc. 22 nd IAEA Fusion Energy Conference, Geneva, Switzerland (2008)).*

Preprint of Paper to be submitted for publication in
Review of Scientific Instuments

ABSTRACT.
In the last years cameras have become increasingly common tools in scientific applications. They are now quite systematically used in Magnetic Confinement Fusion, to the point that infrared imaging is starting to be used systematically for real-time machine protection in major devices. However, in order to guarantee that the control system can always react rapidly in case of critical situations, the time required for the processing of the images must be as predictable as possible. The approach described in this paper combines the new computational paradigm of Cellular Nonlinear Networks (CNN) with Field-Programmable Gate Arrays (FPGAs) and has been tested in an application for the detection of hot spots on the plasma facing components in JET. The developed system is able to perform real-time hot spot recognition, by processing the image stream captured by JET wide angle infrared camera, with the guarantee that computational time is constant and deterministic. The statistical results obtained from a quite extensive set of examples show that this solution approximates very well an ad-hoc serial software algorithm, with no false or missed alarms and an almost perfect overlapping of alarm intervals. The computational time can be reduced to a millisecond time scale for 8-bit 496x560-sized images. Moreover, in our implementation, the computational time, besides being deterministic, is practically independent of the number of iterations performed by the CNN - unlike software CNN implementations.

1. INTRODUCTION

The continuous progress in camera technologies has resulted in commercial products which can be easily operated and have performance which are appealing to many scientific applications. In magnetic confinement nuclear fusion the number of cameras deployed on the various experiments has increased steadily in the last decades. Nowadays they are routine diagnostics to gather information about the plasma wall interactions and various plasma phenomena. Since high temperature plasmas do not emit infrared radiation, IR cameras are very useful tools to determine the surface temperature of the plasma facing components.

The capability of present day materials to withstand the power loads induced by thermonuclear plasmas remains one of the major issues investigated by present day Tokamaks in the perspective of a fusion reactor. This problem, very significant for ITER, is already important on JET and will constitute one of the central aspects of both the operation and the scientific activity after the installation of the new Be wall and the W divertor [1]. Therefore developing reliable infrared thermography diagnostics to determine the surface temperature of the plasma facing components in the main chamber and in the divertor is a very significant issue for JET future programme. New image processing techniques, to provide the necessary information for the operation of the device, has also become a very interesting field of investigation.

On JET the most interesting InfraRed (IR) camera for the development of real time algorithms is installed on a dedicated endoscope providing a wide-angle view (field of view of 70 degrees) in the infrared range (3.5 to 5μm). The wide angle view of the system includes the main chamber and divertor [2]. The diagnostic consists of an endoscope formed by a tube holding the front head mirrors, a Cassegrain telescope, and a relay group of lenses, connected to the camera body. To increase the reactor-relevance of the project, mainly reflective optical components have been chosen, since they can better cope with high neutron radiation. The global transmission of the endoscope is higher than 60% and the diagnostic is designed to measure from the JET typical operating temperature of 200∞C up to a maximum temperature of 2000∞C. The diagnostic spatial resolution is diffraction limited and, assuming a 10% error in the measured photon flux, an overall spatial resolution of 2 cm at three meters has been estimated. A frame rate of 100 Hz at full image size can be achieved and it can be increased up to 10 kHz by reducing the image size to 128x8 pixels, located on any position in the field of view. A typical frame acquired by JET wide angle camera is shown in Figure 1.

The kind of image processing required for hot spot recognition can in principle be performed by dedicated software codes implementing suitable algorithms on serial machines. On the other hand, traditional software solutions are not always completely satisfactory from the point of view of real time applications. Their major problems are due to the sequential nature of the execution flow. This leads to longer computation times and to non deterministic delay since the more complex the image to be processed is, the longer it takes for the traditional software algorithms to analyse one frame. This is not an ideal situation for real time applications, because there is the risk that, when the situation becomes more critical, the response time of the algorithms is less satisfactory. Parallel systems (e.g., multi-processor computers or computer clusters) can help to reduce computational times and improve consistency, by dividing the image into parts to be analyzed by different processors. However, the degree of parallelization is often limited: for example, even if the image is divided into independently-processable parts, the filters have still to be applied sequentially, which means having to wait for the completion of the previous processing step before starting with the following one. Moreover, the algorithms to be applied must be intrinsically parallelizable - not all algorithms are. For these reasons, a hardware system for image processing intrinsically designed to be parallel can be advantageous, since it can typically be executed in shorter and more predictable computation times. The problem with hardware devices is that they obviously are less flexible than software programs, and therefore much more application-dependent; because it is difficult to find a single technique which is adaptable to many application fields, hardware devices have limited applications. However, recent developments in Field-Programmable Gate Arrays (FPGA) technology have opened new possibilities to hardware designers. FPGAs are programmable digital hardware devices, which combine the advantages of hardware solutions (speed, parallelism) with the flexibility of reprogrammability. Of course, the drawback is that performances (in terms of clock frequency) are lower than application-specific integrated circuits, because the latter are optimized for the task they have to accomplish. Nevertheless, the results reported in the following show that FPGAs, with appropriate hardware architectures, can process image streams at frame rates higher than 100 Hz even in the case of complex images - such as the 496x560-sized pictures taken from JET wide angle infrared camera.

In the approach described in this paper, a recently developed computational model, which is very appropriate for generic image processing and is called Cellular Nonlinear Network (CNN), has been implemented using FPGAs. A CNN is basically a matrix of differential equations modelling the evolution of the state of each matrix cell [3]. Each state s value depends on the values of the states in its neighbourhood (usually, a 3x3 submatrix surrounding the target pixel); this makes the CNN a powerful tool for local image processing, provided that state values are mapped on pixel values. An important factor in our choice of using CNNs is that the task accomplished by the CNN (i.e., the filter to be applied to the image) is specified by two 3x3 matrices and a constant (altogether being referred to as a *template*; see section 2) Therefore, it is possible to easily change the kind of filter implemented by the CNN just by modifying those coefficients.

The CNN which has been implemented on our FPGA is based on the Falcon architecture [4], which allows for easy parallelization and overlapped execution of subsequent filters, which renders processing times almost independent of the number of filters applied.

The application of the designed system is the recognition of hot spots using the images collected by JET wide angle IR camera. This paper describes in details how the CNN has been implemented using FPGAs, the hot spot recognition algorithm and its performance, which clearly exceed the results of specific software solutions implemented with the C++ language.

In the remainder of this paper, we first describe the details of the implementation in section 2, explaining the mathematical model and applications of CNNs, what FPGAs are, why they are suitable for CNN implementation and the hardware design which carries out the actual computation. Later, in section 3, the specific algorithm used for hot spot detection, in terms of sequence of CNN templates, is presented in detail. The results obtained are compared with a reference software algorithm for hot-spot detection, in order to evaluate the correctness of the CNN results (see section 4).

Finally, a mathematical analysis of the computation time of the FPGA implementation is provided and compared with classic software algorithms and with software CNN implementations (section 5). Some conclusions are drawn in the last section of the paper.

2. CELLULAR NONLINEAR NETWORKS AND THEIR IMPLEMENTATION USING FPGAS

2.1. THE CNN ARCHITECTURE

A Cellular Nonlinear Network is a rectangular cell array *C(i,j)* (see Figure 2), where each cell is modelled by a nonlinear dynamic system, defined mathematically by state equation, output equation and boundary conditions:

- State equation:

The most generic state equation can be written:

$$\dot{x}_{ij} = -x_{ij} + \sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)y_{kl} + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl} + z_{ij} \qquad (1)$$

where *$S_r(i,j)$* is the set of neighbours of cell *C(i,j)* - usually, the cells within a 3x3 or 5x5 square centred in *C(i,j)*; x_{ij} is the state of the cell; *A(i,j;k,l)* and *B(i,j;kl)* are called respectively the *feedback* and *input synaptic operators*; y_{kl} and u_{kl} are the output and input of cell *C(i,j)*; z_{ij} is a bias constant. However, the most commonly used model is the space-invariant CNN, where the *A* and *B* operators do not depend on the *(i,j)* position, they are simply 3x3 or 5x5 square matrices and the associated operation is a simple multiplication between the correspondent values in the A or B matrices and the *$S_r(i,j)$* neighbourhood of states or inputs - *A*, *B* and *$S_r(i,j)$* having the same size.

Therefore, for 3x3-sized matrices and indexing *A* s and *B* s rows and column from -1 to 1, equation (1) becomes:

$$\dot{x}_{ij} = -x_{ij} + \sum_{k=-1}^{1}\sum_{l=-1}^{1} a_{kl} y_{i+k,j+l} + \sum_{k=-1}^{1}\sum_{l=-1}^{1} b_{kl} u_{i+k,j+l} + z_{ij} \qquad (2)$$

- Output equation

$$y_{ij} = \frac{1}{2}\left|x_{ij} + 1\right| - \frac{1}{2}\left|x_{ij} - 1\right| \qquad (3)$$

This is called standard nonlinearity. The relationship between x_{ij} and y_{ij} is shown in Figure 3.

Boundary conditions: This is the set of rules for assigning values to *virtual cells* - cells which are located beyond the borders of the array (shown in Figure 2 for 3x3 templates). The need for boundary conditions comes from the fact that the 3x3 neighbourhood of border cells is partly located outside of the cell matrix (for example, if we consider the top-left cell, all left and top elements of the 3x3 neighbourhood cannot be mapped onto actual cells). Thus, rules for assigning values to virtual cells are necessary. A common one is the so-called *zero-flux boundary condition*: each virtual cell is created by replicating the closest real cell, in order to avoid sudden value variations which some algorithms might misinterpret.

The choice of *A* and *B* values determines how the state evolves - in other words, determines what the CNN can accomplish.

In a discrete implementation, the state equation is solved by forward Euler iteration:

$$\begin{aligned} x_{ij}(n+1) &= (1-h)x_{ij}(n) + \\ &+ h\left(\sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)y_{kl} + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl} + z_{ij}\right) \end{aligned} \qquad (4)$$

where the h time step value models the flow of time at each iteration.

Although CNNs can be applied to several fields, not necessarily scientific, they were specifically designed to perform real-time ultra-high (more than 10.000 frames per second) frame rate image processing. After all, the bidimensional cell layout can be naturally mapped to an image matrix, by associating each cell state to the corresponding pixel s grayscale value; besides, it is possible to convert classic image processing filters to *(A,B,z)* triples (in fact, there already exist several CNN templates libraries).

As a last mathematical remark, it has been proved [3] that under the following three conditions:
- *A* and *B* being linear and memory-less operators (that is, $A_{ij}\sum x_{ij}$ is a scalar multiplication);
- *u(t)* and $z_{ij}(t)$ being continuous functions of time;
- The output function $f(x_{ij})$ being a Lipschitz-continuous function, the CNN evolves to a one and unique solution. In our model, these requirements are easily fulfilled, since the *A* and *B* operators perform simple multiplications between their coefficients and the instantaneous value of state and input *u(t)* and *z(t)* are supposed to be constant, because we assume the CNN is designed for static image processing; the output function is the standard nonlinearity function, which is limited and therefore Lipschitz-continuous.

Finally, it is possible to prove [3] that, if provided with a memory where to keep temporary results, the CNN is equivalent to a Turing Machine, in the sense that it can implement any algorithm that works on a finite set of values. Therefore a CNN with a memory has the same computational capability as a serial machine of the Von Neumann type. Of course the main competitive advantage, very interesting for applications such as image processing, is the intrinsic parallelism.

2.2. THE FPGA TECHNOLOGY

The basis for FPGA technology was first introduced by Xilinx in 1985, under the name of Logic Cell Array (LCA). That architecture, which since then has been constantly improved and extended, consisted of an array of relatively simple functional blocks, interconnected by a network of programmable switches for input-output redirection; however, though the blocks were very simple, programmable interconnections allowed the implementation of complex applications. Present day FPGA architectures are usually classified according to logic-block granularity and routing architecture.

Coarse-grained FPGAs use high-functionality blocks, whose large number of inputs on the other hand require greater routing resources; whereas fine-grained FPGAs are made of very simple blocks, which allow for better block resource usage, but require more die space for the interconnection network. The routing architecture (which strongly influences performances, since signal propagation delays can be very high if the interconnections are poor) defines three classes of FPGAs: row-based FPGAs, symmetric FPGAs and cellular FPGAs, which differ from each other by the type of logic block they use and by the predictability of net delays. However, independently of their internal architectures, FPGAs block arrays are always surrounded by a layer of input/output blocks, which allow for communication with external devices. Xilinx Virtex-family FPGAs, which are the ones used for the implementation described in this paper , provides I/O blocks supporting 16 standards such as LVTTL, AGP, PCI, HSTL and SSTL [5]. Moreover, another reason why we decided to use Virtex-family devices, and specifically the XC4VSX35 one, is the presence of a high number (192 on the device used) of so-called XtremeDSP blocks, each of which contains a 18x18 bit multiplier, an adder and an accumulator. Such blocks greatly improve performances and chip-usage of computation-bound application, such as image processing.

Of course, one of the main advantages of FPGAs is the re-programmability, which makes them a very flexible tool for implementing modules with variable architecture and internal structure, such as a CNN, one of whose best features is the capability of easily adapting to the data to be processed and to the algorithm to be run. Xilinx provides a set of useful software and hardware design tools, such as the IP Core Generator, a software utility which creates netlists for common hardware modules (for example, shift registers, RAMs, etc) and the System Generator, a Matlab Simulink extension for graphical hardware design (even though the CNN core was written in VHDL, we used System Generator blocksets for interfacing the CNN black box with the external world) and for hardware/software cosimulations (in which the computation is divided between Matlab - which transfers the test images to the FPGA - and the FPGA itself - which performs the actual calculations).

2.3. THE CNN IMPLEMENTATION USING FPGAS

Our CNN implementation on FPGAs is based on the Falcon architecture [4], which in turn extends the Castle architecture [3] (originally developed by CNN pioneers Leon Chua and Tam s Roska) by making it possible, thanks to the FPGA flexibility, to configure the bit width of state and template values, the width of the CNN cell array, the number of stored templates and the number and arrangement of processor cores (which will be described in detail later). In our implementation, the state value of the cell is equal to its output value and is limited in the [-1, 1] range, according to the so-called Full Signal Range (FSR) model.

Thus, the discrete-time state equation becomes:

$$\begin{aligned} x_{ij}(n+1) &= (1-h)x_{ij}(n) + \\ &+ h\left(\sum_{C(k,l)\in S_r(i,j)} A(i,j;k,l)x_{kl}(n) + \sum_{C(k,l)\in S_r(i,j)} B(i,j;k,l)u_{kl}(n) + z_{ij} \right) \end{aligned} \tag{5}$$

In order to reduce the number of calculations, the *A* and *B* template matrices are modified so that they intrinsically include the time step value. The modified template matrices become:

$$\begin{aligned} A' &= \begin{pmatrix} ha_{-1,-1} & ha_{-1,0} & ha_{-1,1} \\ ha_{0,-1} & h(a_{0,0}-1) & ha_{0,1} \\ ha_{1,-1} & ha_{1,0} & ha_{1,1} \end{pmatrix} \\ B' &= hB \end{aligned} \tag{6}$$

and the state equation can be split into two parts:

$$\begin{aligned} x_{ij}(n+1) &= x_{ij}(n) + \sum_{C(k,l)\in S_r(i,j)} A'(i,j;k,l)x_{kl}(n) + g_{ij} \\ g_{ij} &= \sum_{C(k,l)\in S_r(i,j)} B'(i,j;k,l)u_{kl}(n) + z_{ij} \end{aligned} \tag{7}$$

The main advantage of this solution is that, if the input is constant (as it is in our application, since we process images one by one), then the g_{ij} value is also constant and can be computed only once, at the beginning of the execution of the template, greatly reducing the computation load for the following iterations.

The Falcon architecture is basically an array of processing cores. The input image is divided among the array s columns, in order to increase parallelism, whereas each row performs an algorithm iteration (see Figure 4) - that is, in order to iterate the execution of a template ten times, ten core rows would be needed, independently of the number of columns. The simplest arrangement contains one column (that is, the image is not divided into pieces to be independently processed) and a number of rows equals to the number of desired iterations.

The basic computing unit is called a *core*, and it executes a full iteration of the state equation. A core takes as input the sequence of states from the past iteration (or the template s initial state), the sequence of g_{ij} constants associated to the corresponding states and a start-in signal, which notifies the beginning of the computation. A core outputs the newly evaluated states, the corresponding constants (unchanged) and a start-out signal. All cores in each row are synchronized by a set of control units (which implement finite-state machines), each of which controls a specific part of the computation.

The internal architecture of a core is shown in Figure 5:

The memory unit contains three shift registers, whose size is equal to the width of the input image. Inputs serially enter the upper-most shift register and traverse all shift registers, and at each moment, the outputs of the shift registers contain a set of three vertical pixels, which are then passed on to the mixer unit.

The mixer unit takes three states at a time from the memory unit and arranges them for the arithmetic unit (which does the actual computation), in such a way that every three clock cycles the arithmetic unit has all the values it needs to compute a new state.

The arithmetic unit simply performs the computation of the new state value, according to the modified discrete-time state equation (7).

3. THE HOT-SPOT RECOGNITION ALGORITHM IMPLEMENTED USING THE CNN

The implementation of the CNN architecture with FPGA, described in the previous section, has been applied to the issue of hot spot detection in JET. The images analysed have been collected with JET wide angle infrared camera.

The implementation of a hot spot detector with the CNN consists basically of finding a sequence of templates which can identify the critical regions inside the camera field of view with sufficient reliability. With regard to the algorithm, which has been developed to identify the hot spots, at first the input image is thresholded according to temperature parameters, which can be defined by the user depending on their needs. Thanks to the camera calibration data, which allows a temperature-grayscale value mapping, the grayscale input can be converted to a temperature matrix. A first threshold is applied so that the pixels with temperature lower than 500∞C are blackened. A second threshold is applied to the region of the image where the divertor is located and blackens all pixels with temperature lower than 800∞C (since the divertor is designed to tolerate higher temperatures than the rest of the vacuum chamber). Since thresholding assigns a pixel s new values according to a static rule and depending only on the pixel s current value, there is no need to apply a CNN template for this step. Figure 6 shows the output of the threshold filter.

After thresholding, the binary image contains white pixels where the temperature is over the safety threshold. However, the presence of white pixels does not necessarily imply the presence of a full-blown hot spot. It is necessary to check the size of the hot region and how long the region remains above the critical temperature, before considering it a hot spot worth triggering an alarm. Another problem is that several small non-contiguous regions might actually belong to the same hot spot if they are close enough.

The first template applied is a variation of the *PointRemoval* [6] template. The original template removes isolated black pixels; we modified the z_{ij} parameter so that the template actually removes all black pixels which have less than two black neighbours. This template is applied because isolated pixels (or isolated pairs of contiguous pixels), though they are generally negligible for hot spot discrimination, can cause problems to the following templates. Only one iteration of this template is executed.

The second template applied is a morphological operator [8] called *DirectedGrowingShadow* [6]. Basically, it increases the size of the object by creating a shadow which can be directed in any desired direction. In our case, we tuned the template s coefficients so that the object is expanded more horizontally than vertically, since we empirically found that this choice allows a better detection of hot spots on the inner limiter of JET vacuum chamber. This template is executed for six iterations. The application of this template allows merging objects which are sufficiently close into a single bigger object. The main problem with this template (and, in general, with all morphological operators which increase the size of an object) is that it modifies the original image by adding virtual pixels, which have no physical correspondence in the input image. However, this approach, if properly managed, provides acceptable results, since in the worst case it will detect false alarms, but never miss any.

The third template is called *ConcaveFiller* [6], which fills object cavities with black pixels. The purpose of this template is to avoid that the following shrinking phase might separate previously-unified objects. The CNN performs four iterations of this template.

The fourth template being applied has the task of reducing objects, taking their size back to the original (that is, before it was increased) scale. Actually, the template name is *ObjectIncreasing* [6]; however, the template is designed to consider an object as a set of black pixels. Since our objects are actually white, the template will consider them as holes to be filled (the only real object being the black background), which has the effect of reducing the size of the hot regions. Two iterations of this template are performed, which approximately compensates for the effects of *DirectedGrowingShadow*.

After these four templates, the output image consists of a set of contiguous objects representing possible distinct hot spots. At this point a last template called *SmallObjectRemover* [6] is applied. It allows discriminating between regions according to their size, so that only regions which are too small are removed. The number of iterations performed by this template can be chosen according to the desired degree of safety. For example, the optimal number of iterations (that is, the one which minimizes the difference between the reference algorithm and the CNN algorithm) is ten. However, it is possible to tune this parameter so that the discrimination criterion is less strict (e.g., one could lower the number of *SmallObjectRemover* iterations); this might lead to the detection of false alarms, but guarantees that no real alarms are missed. Section 4 shows statistical results about two possible choices of the setting of the number of *SmallObjectRemover* iterations.

After this template sequence has been executed, searching for hot spots is just a matter of checking whether there are still white regions in the output image. Figure 7 shows the output image for each template.

The described algorithm allows verifying whether a single image contains hot-spot-like regions. However, for such a region to be actually a hot spot, it is required that it persists for a certain time - not for just a frame. The problem is that a CNN like the one used is only able to process an image at a time - in this sense, it is without memory. The solution to this problem is based on a simple consideration: if a pixel belongs to a hot spot, then, in our thresholded image, it will stay white for a certain number of consecutive (say, N) frames; therefore, the mean of the grayscale intensity value for that pixel over the last N frames will have to be above a certain threshold. If a region becomes white for just a single frame, than the mean value of the pixels in that region over the last N frames (with N being sufficiently large) will be under the threshold. So what is actually provided as input to the CNN is not a single frame, but a frame obtained by calculating the mean of the last N (in our implementation, four) frames. If the CNN detects a hot spot on this image, it will mean that a hot region must have persisted for at least the last four frames, which is enough for us to say that it really is a hot spot.
It is worth mentioning that all the described parameters can be easily modified by the user to meet their requirements.

4. PERFORMANCE OF THE DEVELOPED ALGORITHM FOR THE HOT SPOT DETECTION

In order to validate the results of the CNN algorithm, it is necessary to create a suitable database of videos where hot spots are present. To this end, 10 videos of the IR wide angle camera, representative of the typical regimes of JET operation, have been analysed manually. About 12100 frames have been scrutinised with the help of experts, to determine which objects in the images are really hot spots. Moreover, to have a term of comparison for the computational aspects of the CNN implementation on the FPGA, an algorithm has been written in C++ and run on a serial machine to analyse the same videos. The devised algorithm is relatively simple; first a typical thresholding operation is implemented using the same two thresholds used in the CNN implementation. Then a clustering step is applied to the surviving pixels in order to gather those belonging to the same potential hot region. This is the most delicate part of the method since the hot spots can appear in the images as a series of small objects. A specific sorting algorithm has been developed to solve even the most delicate situations. The global properties of each region are then classified in a dynamical array. In this way the evolution of the critical regions can be followed to decide whether a certain object in an image persists long enough to be considered a real hot spot. An example of the obtained results is shown in Figure 8.

The accuracy of this algorithm is very high and the list of hot spots identified is perfectly consistent with the manually-created database. For this reason, we have used these results as reference data for the comparison with the CNN algorithm.

The CNN algorithm described above produced results almost identical to the reference data. In particular, no false alarms or missed alarms have been detected. Table 1 compares the alarms recognized by the reference algorithm to those recognized by the optimal configuration (in the sense that it minimizes the differences from the target results) of the CNN, and Table 2 shows the high degree of coincidence between the two result sets. It is possible to see that the alarm intervals almost perfectly overlap, which proves the quality of the CNN solution in approximating the software algorithm. The performance of the CNN optimised algorithm can therefore be certainly considered more than adequate, in terms of success rate, for the real time detection of hot spots

Pulse	Number of frames	Reference alarms	CNN alarms
65000	800	89-121	90-119
65409	800	100-118 394-424	101-117 392-424
65410	800	98-118	102-118
65411	800	98-120	102-119
65412	800	97-119	100-119
65420	800	98-119	101-119
65430	1600	No alarm	No alarm
66734	1700	204-252	203-254
66866	2000	250-287	250-288
66867	2000	255-289	255-291

Table 1: Comparison between the alarms detected by the reference algorithm and by the optimal CNN algorithm. The numbers in the two left columns identify the frames in which a hto spot has been detected.

Pulse	**Hot-spot overlap ratio**	**Total overlap ratio**
65000	30/33 = 91%	797/800 = 99%
65409	17/19 = 89% 31/33 = 94%	796/800 = 99%
65410	17/21 = 81%	796/800 = 99%
65411	18/23 = 78%	795/800 = 99%
65412	20/23 = 87%	797/800 = 99%
65420	19/22 = 86%	797/800 = 99%
65430	-	1600/1600 = 100%
66734	49/52 = 94%	1697/1700 = 99%
66866	38/39 = 97%	1999/2000 = 99%
66867	35/37 = 95%	1998/2000 = 99%
Total	274/302 = 90.7%	12072/12100 = 99.8%

Table 2: Statistics on the overlapping of the alarm intervals detected by the reference algorithm and by the optimal CNN algorithm. The numbers in the two left columns identify the frames in which a hto spot has been detected.

As we said in section 3, it is possible to tune the CNN algorithm so that the hot spot discrimination criterion is less strict, which leads to earlier (in terms of frame number) detection of hot spots, but increases the chance of false alarms. Table 3 and Table 4 show the results obtained by lowering the number of *SmallObjectRemover* iterations from ten to eight. You can see that the detected alarm intervals are in general slightly wider than the reference ones, and. in most cases. even more precise than the ten-iteration configuration; however, for pulse 66734, the alarm detection begins twenty frames earlier than the actual alarm interval, which sensibly influences the overall statistics, and, most importantly, may be enough to consider it a false alarm.

Pulse	**Number of frames**	**Reference alarms**	**CNN alarms**
65000	800	89-121	88-121
65409	800	100-118 394-424	99-118 392-426
65410	800	98-118	98-118
65411	800	98-120	97-119
65412	800	97-119	96-119
65420	800	98-119	95-119
65430	1600	No alarm	No alarm
66734	1700	204-252	183-254
66866	2000	250-287	250-288
66867	2000	255-289	255-291

Table 3: Comparison between the alarms detected by the reference algorithm and by the "safer" version of the CNN algorithm.

Pulse	Hot-spot overlap ratio	Total overlap ratio
65000	33/34 = 97%	799/800 = 99%
65409	19/20 = 95% 31/35 = 89%	795/800 = 99%
65410	21/21 = 100%	796/800 = 99%
65411	22/24 = 92%	798/800 = 99%
65412	23/24 = 96%	799/800 = 99%
65420	22/25 = 88%	797/800 = 99%
65430	-	1600/1600 = 100%
66734	49/72 = 68%	1677/1700 = 99%
66866	38/39 = 97%	1999/2000 = 98%
66867	35/37 = 95%	1998/2000 = 98%
Total	293/331 = 88.5%	12059/12100 = 99.6%

Table 4: Statistics on the overlapping of the alarm intervals detected by the reference algorithm and by the "safer" version of the CNN algorithm.

5. THE COMPUTATION TIME REQUIRED BY THE FPGA IMPLEMENTATION OF THE ALGORITHM COMPARED WITH OTHER SOLUTIONS

The algorithm in C++ to be run on a serial machine presents very good performance in terms of accuracy in detecting the hot spots.The main drawback of the serial algorithm is its strong dependence of its computational time on the contents of the images and in particular on the number of pixels above threshold to be processed. This is summarized in Figure 9, in which the computational time is plotted versus the number of processed pixels.

This strong dependence can disrupt the operation of the algorithm as is shown in the example of Figure 10.

The implementation of a hot spot recognition algorithm by means of CNNs has two main advantages over a software implementation using traditional sequential CPUs or DSPs: independence of the computational time from the image content and therefore deterministic computational times.
- The content of the image does not influence the computation time. This is because an algorithm implemented with a CNN consists of the iterative application of the same equation. The actual values to be computed do not have any influence on the speed of the final calculation. On the contrary, software algorithms analyze the content of the image in order to find interesting regions, which makes computation time dependent on the given input. A CNN algorithm in a certain sense transfers complexity from the algorithm itself (which, for CNNs, is always the same) to templates - or rather, to the choice of templates.
- Computation times are deterministic. This is a direct implication of the previous considerations. Since the CNN iteration algorithm does not depend on the data to be processed, it is possible to calculate execution times offline. Of course, a software algorithm can be designed and optimized according to the task it has to accomplish, while generally the CNN is a sort of approximation of an ad-hoc algorithm. Nevertheless, CNNs have proved to be suitable for image processing, thanks to

the fact that their operators (templates) can be customized to perform almost any task.

As far as the comparison between CNN on FPGAs versus implementations on serial computers is concerned, the advantages of the FPGA choice are a bit more subtle. This is because new computation-oriented processors (such as DSPs) have been developed, which allow for optimization of memory usage and access and quick execution of operations (such as multiplications) that would require several clock cycles on general-purpose processors. However, our time estimation, presented in the following, indicates that a CNN implementation on FPGA is actually faster than a DSP implementation.

First of all, in our design, a single processing core is able to compute a new state value every three clock cycles, which is quite difficult to obtain with a DSP, not only because of the number of operations to be performed in order to compute new state values, but also because memory accesses require a considerable amount of time, whereas the Falcon architecture s memory and mixer units allow for efficient storage and presentation of values to the arithmetic unit. However, DSPs can usually reach a higher clock frequency than FPGAs; so, for the sake of argument, let us suppose this compensates for the time required of computing new states. Even with this assumption, there are two more important factors in favour of the FPGA choice.

First of all, the core-array architecture makes it easy to add new core columns in order to divide processing among them. If the execution of the CNN algorithm on the full image on a single-column array takes T seconds, the execution on C columns would take T/C seconds. With an array of DSPs it is difficult to obtain the same reduction in computational time, because processors would need to concurrently access the memory (which causes longer waiting intervals) and to communicate between each other - all things that on an FPGA require no time, since the hardware architecture can be (and has been) designed to perform inter-core communication during normal computation. Moreover, as far as costs are concerned, adding N core columns is just a matter of changing a VHDL design file, while adding N DSPs requires additional hardware.

The second advantage of FPGA-CNNs over DSP-CNNs is probably even more important than the previous one. Let us suppose the image to be processed is 496x560 pixels in size like the case of JET IR camera. In terms of clock cycles, the time required for a CNN iteration on our architecture can be computed as the sum of the two terms T_t and T_c which are defined as:
- T_t (transient), the time needed for the first output to be made available at the output by the arithmetic unit
- T_c (computation), the time needed for computing all following new state values, that is $3 \cdot 496 \cdot 560 \approx 10^6$ (the $3 \cdot$ factor is there because every new value requires three clock cycles to be computed).
T_t time is computed as the sum of the following values:
- The time required for filling the memory shift registers. At the beginning of the computation, the first two shift registers are both filled with the first line, in order to satisfy the zero-flux boundary conditions, so this time equals $3 \cdot 2 \cdot 496 \approx 3000$;- The time required for the mixer unit to arrange the first inputs to the arithmetic unit, which is 9 clock cycles;

- The time required for the arithmetic unit to compute the first state value, which is about 20 clock cycles (the multipliers latency is set to 18 clock cycles, which is a very high value, but allows for modification of state width without having to change the timing control units).

Overall, the T_t time is about 3000 clock cycles, which is very short compared to T_c.

Even if we suppose that a CNN software implementation can run an iteration as fast as the CNN can, there is still a major difference which makes the FPGA implementation much better: As soon as the first output is ready from the arithmetic unit, the next core row can begin to process it. A DSP has no way to do this, because its execution flow is strictly sequential, so it has to wait until the previous iteration has been completed before beginning the new one. This means that if an algorithm requires 25 iterations (a value comparable to the case of the algorithm for the hot spot identification), the FPGA implementation can run it in $25 \cdot T_t + T_c$ (that is, each iteration has to wait only until the first output of the previous iteration is evaluated), whereas the DSP implementation would take $25 \cdot T_c$, which is about 25 times the FPGA computation time, since T_t is negligible compared to T_c.

Finally, some considerations about the processing performance of the algorithm. Our design was synthesized for a Virtex-4 XC4VSX35 FPGA mounted on an ML402 evaluation board. The board provides an external oscillator socket and a built-in 100MHz oscillator, that latter being that which we used as clock signal to the FPGA. We have already estimated the number of clock cycles required to process a single image, which is around 10^6. A million clock cycles at 100MHz means $10^6 \cdot 10^{-8}\, s = 10^{-2}\, s = 10\, ms$. So, the maximum input frame rate is 100 Hz, which is already a very high value for most cameras. Moreover, the previous calculations assume a one-column core array, that is, no parallelism. As we said earlier, if we divide the image among N columns, the computation time is exactly reduced by N times, which means the maximum input frame rate rises up to $N \cdot 100$Hz. For example, with ten columns it would be possible to process 1kHz image streams, and of course, it is still possible to add more columns to the core array - the upper limit being the size of the FPGA chip.

CONCLUSIONS

The evolution of camera technology has made them a common tool in Magnetic Confinement Fusion, and in our work we have analysed a system for real-time image processing, based on an implementation of the Cellular Nonlinear Network paradigm on Field Programmable Gate Arrays devices, and tested it on a hot spot recognition application for JET. The statistical comparison between a software recognition algorithm, whose results have been taken as reference data, and the CNN algorithm shows the accuracy of this solution, which provides results that are almost identical to the target ones.

The results show that the CNN can approximate very well traditional software algorithms for hot spot recognition, with the considerable advantage of the processing time being independent of the content of the image and deterministically computable.

The deployment of the algorithm on FPGA devices has the advantage of hardware implementation (which provides fast execution and parallelism) and flexibility, thanks to the reprogrammability feature of FPGAs, which allows the adaptation of the CNN algorithm to the specific application. Finally, we have shown a performance comparison involving classic software algorithms and DSP CNN implementations; the FPGA implementation of the CNN turns out to be superior to both, thanks to the aforementioned advantages of CNNs over classic algorithms and to the enhanced parallelization ability of the FPGA with respect to DSPs.

References

[1]. J.Pamela et al, Journal of Nuclear Materials **363—365**(2007) 1—**1**

[2]. E.Gauthier et al. Fusion Engineering and Design, Volume **82**, Issues 5-14, October 2007, Pages 1335-1340

[3]. L. Chua, T. Roska, Cellular neural networks and visual computing: Foundations and applications , Cambridge University Press, Cambridge, 2004.

[4]. Z. Nagy, P. Szolgay, Configurable Multi-Layer CNN-UM Emulator on FPGA , IEEE Trans. on Circuits and Systems I: Fundamental theory and applications, Vol. **50**, pp. 774-778, 2003.

[5]. Xilinx DS112 Virtex-4 Family Overview datasheet, from the Xilinx website.

[6]. Cellular Sensory Wave Computers Laboratory, Hungarian Academy of Sciences, edited by L. K k, K. Karacs, T. Roska, Cellular wave computing library (templates, algorithms and programs) , 2007, available at http://cnn-technology.itk.ppke.hu/Library_v2.1b.pdf .

[7]. P. Keresztes, A. ZarÆndyT. Roska, P. Szolgay, T. H dv gi, P . J n s, A. Katona, An emulated digital CNN implementation , Int. Journal of VLSI Signal Processing, 1999.

[8]. M.Sonka, V.Hlavac, R.Boyle Image processing, analysis and machine vision Thomson, London, 2008 Third edition

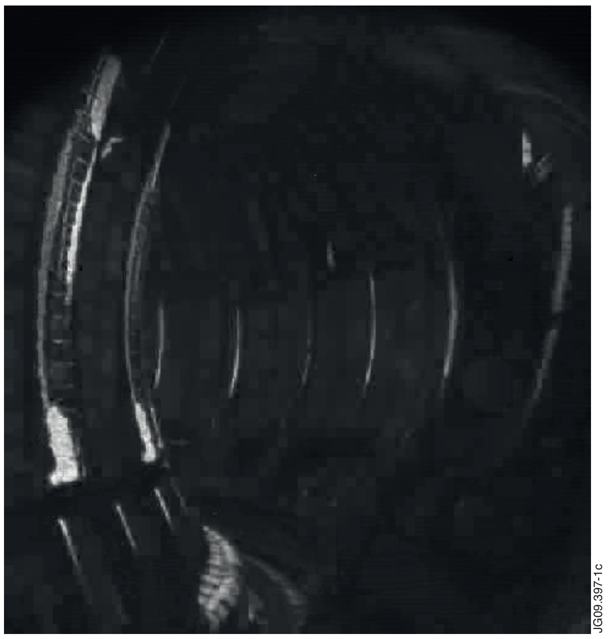


Figure 1: Example of image acquired by JET IR wide angle camera.
The white pixels indicate regions of high surface temperature.

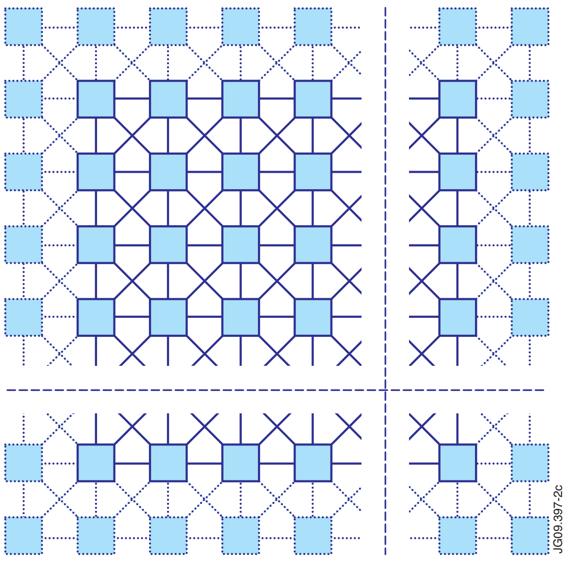


Figure 2: CNN array and virtual cells, shown in a lighter shade of blue.

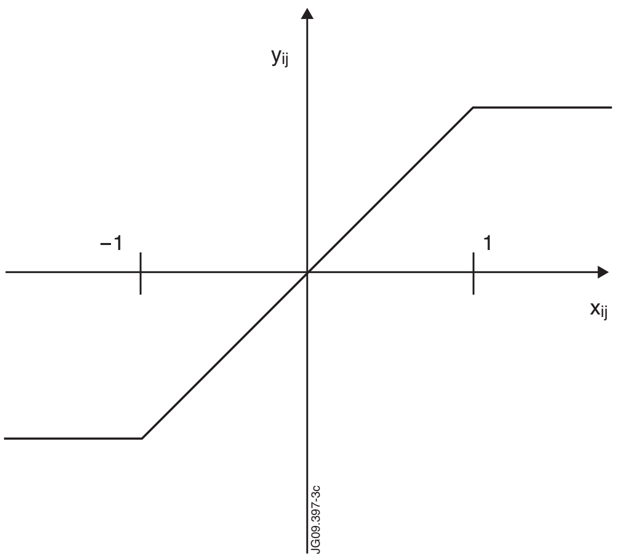


Figure 3: CNN cell output.

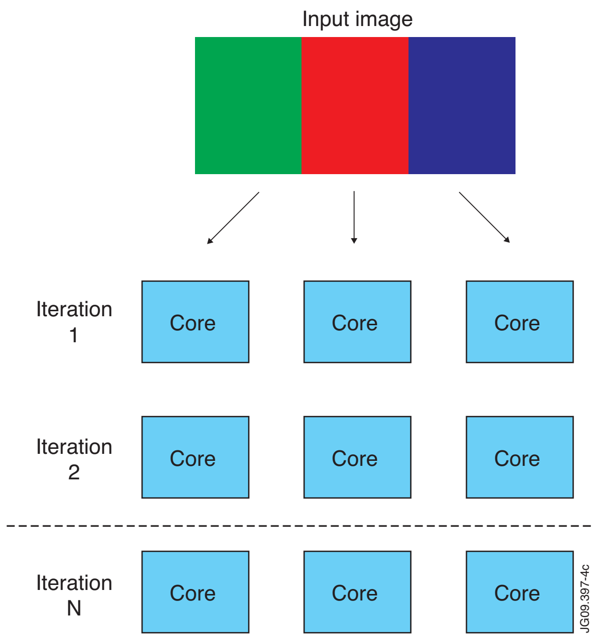


Figure 4: Core array architecture. Image division and iteration chain.

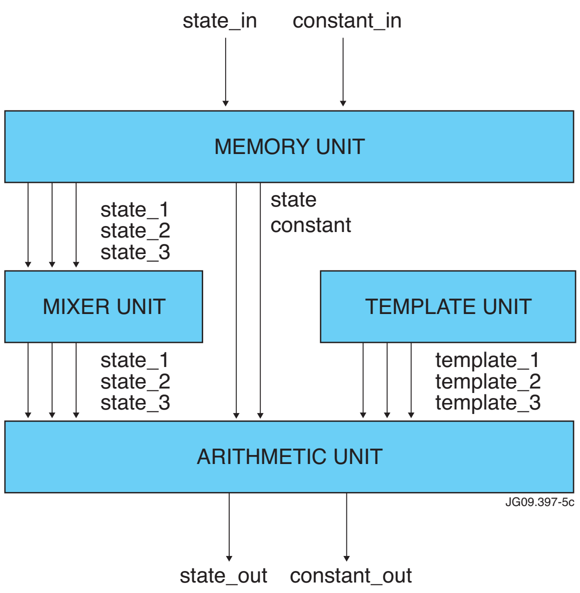


Figure 5: Core architecture.

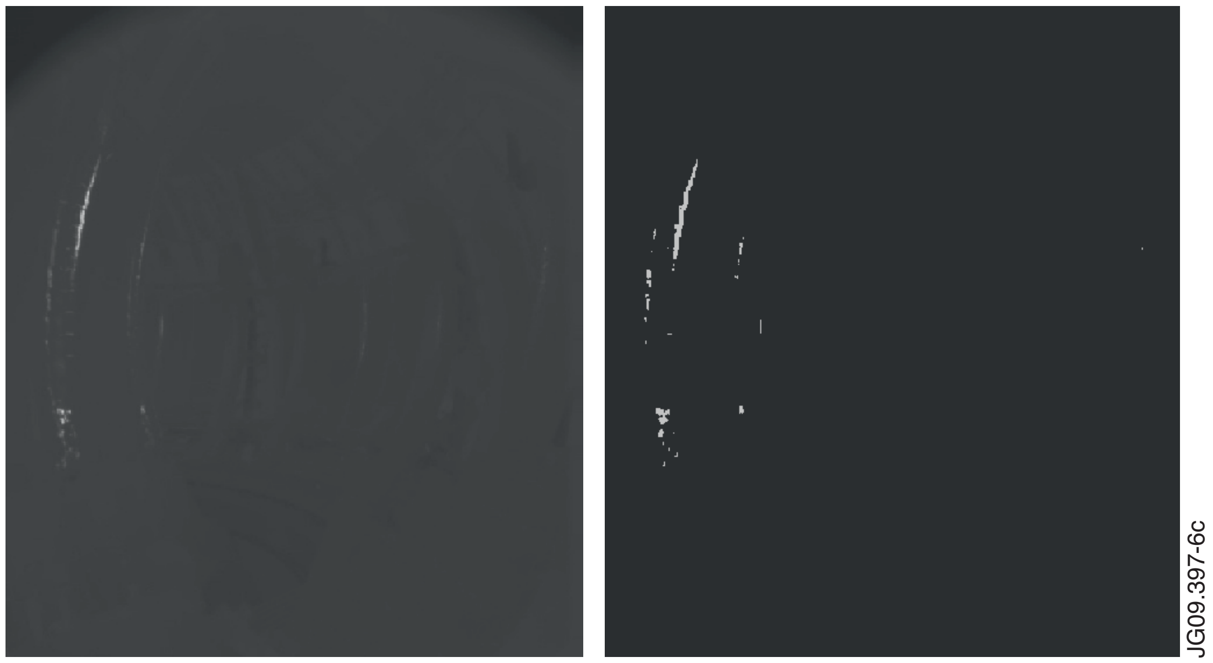


Figure 6: Application of the temperature threshold filter (at 300^{o}C) on an input image.

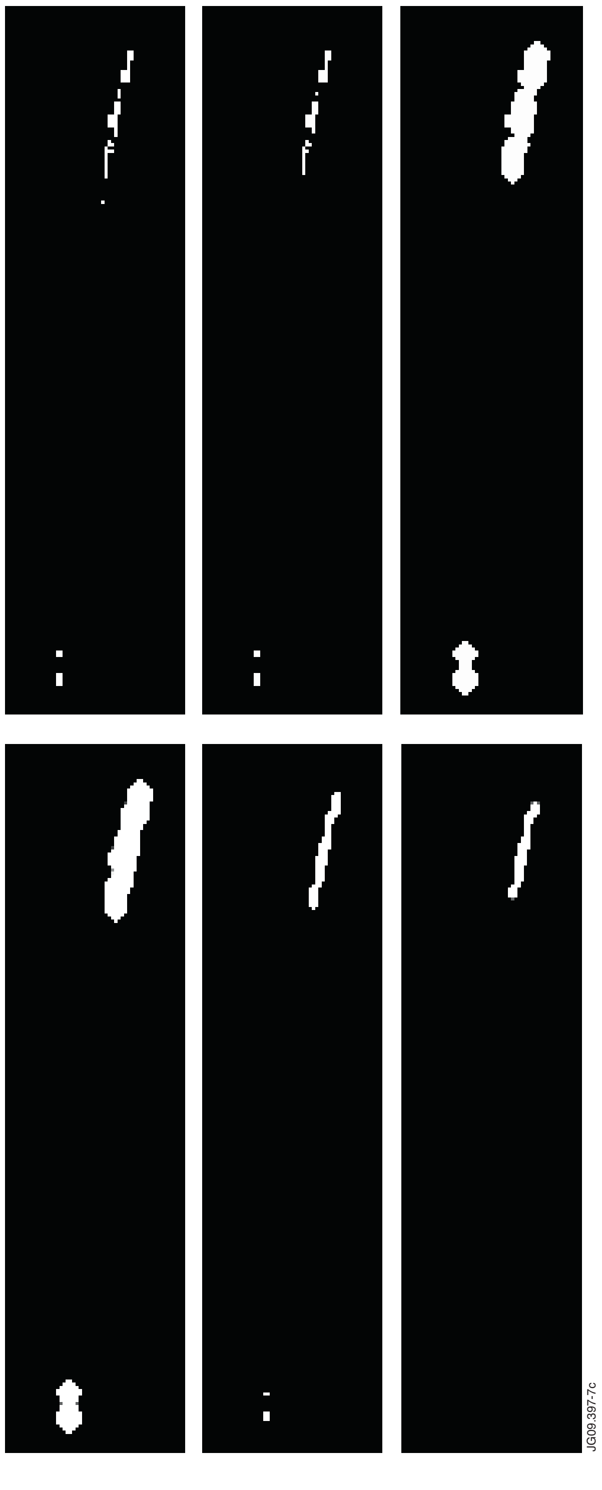
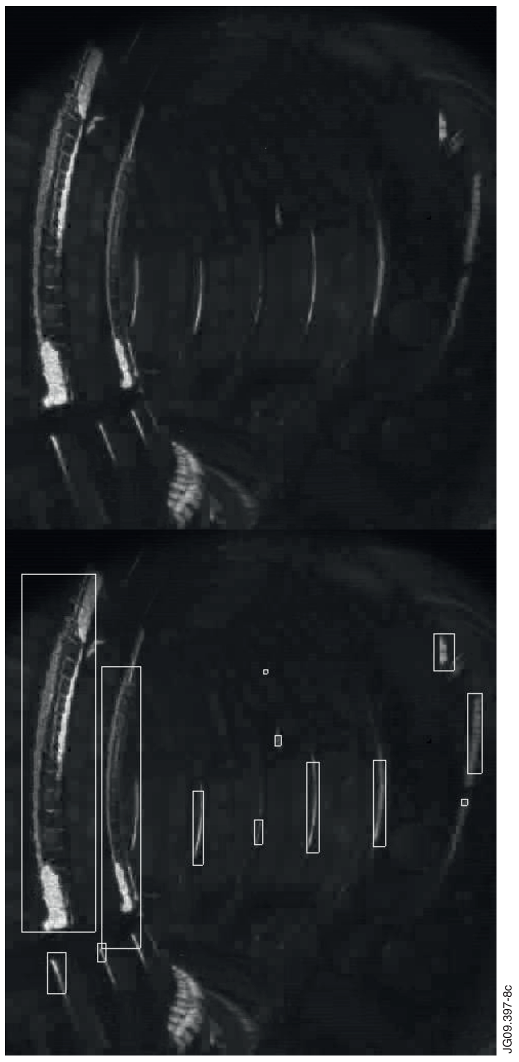


Figure 7: Top-left: Zoom of thresholded input; Top-center: Application of our variant of PointRemoval; Top-right: Application of DirectedGrowingShadow; Bottom-left: Application of ConcaveFiller; Bottom-center: Application of ObjectIncreasing; Bottom-right: Application of SmallObjectRemover.



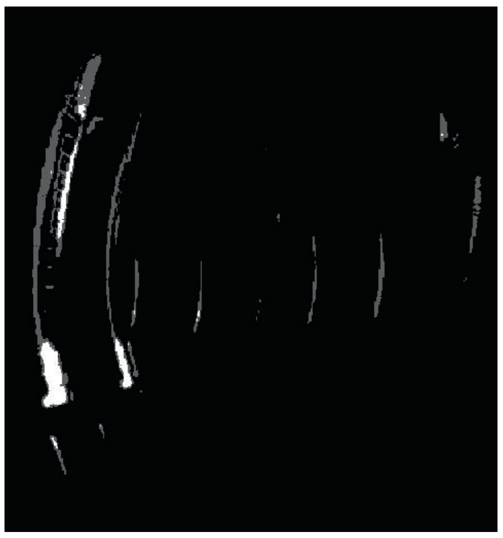


Figure 8: The first image is the original KL7 image; the second one is the threshold image with three different shades of gray: black for pixels whose temperature is lower than the first threshold; grey for pixels higher than the first threshold and white for pixels higher than the second threshold. On the right the final processed image is shown; each rectangle is drawn by the algorithm and represents how the algorithm separates the hot regions.

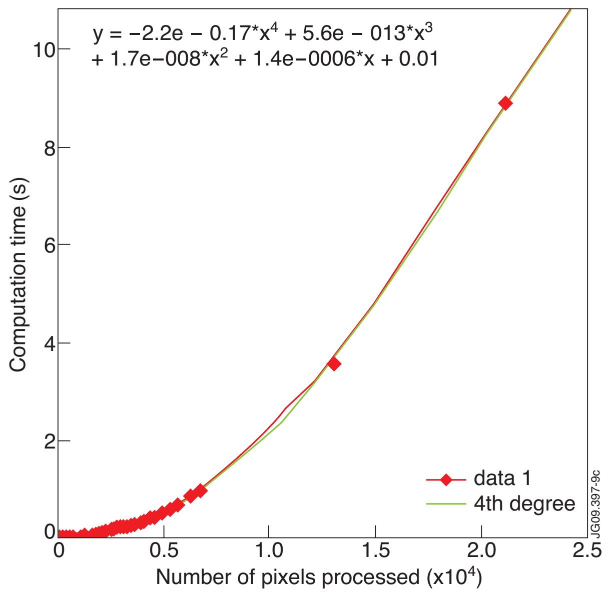


Figure 9: Computational time versus the number of pixel processed by the serial algorithm.

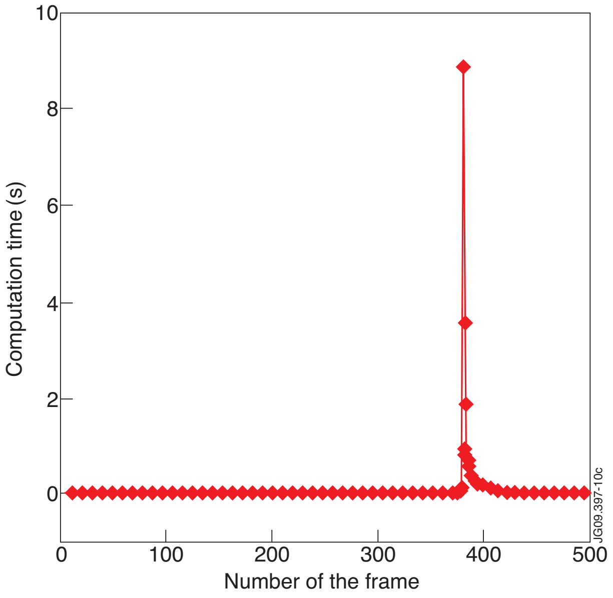


Figure 10: In traditional serial algorithms, problematic frames can make computation time rise and become unacceptable.