Filtering performance optimisation of medical images using Chip Modelling (SystemC)

Henrique VICENTE SOUZA, Graduate Student ENSTA-Paristech,
Alexandre LEFORT, Graduate Student ENSTA-Paristech, and Nicolas VENTROUX, Project Manager CEA LIST

***Abstract*— The fundamental task required for any image processing application is to identify all the features of the image. In this paper Median filter is used to reduce noise, Threshold filter is used to convert grayscale or color pixels into high contrast, and Sobel filter is used to edge detection process. The application of all these filters has normally a high computational cost, so the hardware/software optimisations are implemented using Chip Modelling (SystemC) in a group of images of a patient with cerebral cancer.**

Keywords*— *SystemC, Image Processing, Hardware/Software Optimisation, Embedded Systems.

I. Introduction

The Object of this paper is to present an image processing chip for an MRI system. It will be implemented on the VirtualSoC platform and will use an hardware accelerator. The image processing deals with 126x96 grayscale images of brain sections. The processing has to facilitate tumor detection by giving away its contour. The process includes three steps. First, it will reduce the noise of the picture with a median filter. Then it will use a threshold filter in order to do a segmentation of the picture. Then, a Sobel filter will be applied to show the contour. The time target is fixed to one millisecond for the whole process.

II. First implementation

In this section the methods used in the first implementation of the filters are discussed. First of all, the prior consideration is to write the codes of the Median, Threshold and Sobel filters in C language in order to analyse their average computation time in medical imaging processing.

A. Median Filter

The median filter is normally used to reduce noise in an image, so it considers each pixel in that in turn and looks at its nearby neighbors to decide whether or not it is representative of its surroundings. Instead of simply replacing the pixel value with the *mean* of neighboring pixel values, it replaces it with the *median* of those values. The median is calculated by first sorting all the pixel values from the surrounding neighborhood into numerical order and then replacing the pixel being considered with the middle pixel value.

For the first implementation of the Median filter, a sort algorithm called quicksort can be implemented in C language as shown in Fig. 1.

```
void quicksort(unsigned char *vector, int left, int right) {
   int r;
   if (right > left) {
   r = partition(vector, left, right);
   quicksort(vector, left, r - 1);
   quicksort(vector, r + 1, right);
   }
}
```

Fig. 1: Quick sort implementation with recursive calls

The vector that needs to be sorted has 9 elements corresponding to the current analyzed pixel of the image and its 8 neighbors. After running several tests with Quick Sort algorithm is observed that this recursive method has a high computational cost in image filtering process, requiring an extremely high processing time of about 23 miliseconds.

B. Threshold

The Threshold filter is the most simple and the fastest part of the process. It converts grayscale or color images into high contrast, black and white images. All pixels darker than 100 value are converted to black (0 value).

Once this algorithm is based only on the verification of each image's pixel with the value of a pixel edge (which contains 100 value), the implementation must be performed using two nested loops to read all pixels, defined as an array of unsigned char terms.

C. Sobel Filter

The Sobel Filter is normally used to image edge detection process, in order to locate the edge of an image which by finding the approximate absolute gradient magnitude at each point I of an input grayscale image. The Sobel Filter is sensitive to noise in pictures that effectively highlight them as edges. In the presented paper a Gradient method for filter design is used: it detects the edges by looking for the maximum

Henrique VICENTE SOUZA is an Engineering Student at ENSTA-ParisTech, Paris, France, doing the 2nd year MSc in the advanced specialisation ”System Engineering: Robotics and Embedded Systems” as a Double Degree Student, being at the same time an Undergraduate Student at University of Campinas, Campinas, Brazil, in Automation and Control Engineering, e-mails: henrique.vicente-souza@ensta-paristech.fr, and h105063@dac.unicamp.br.

Alexandre LEFORT is an Engineering Student at ENSTA-ParisTech, Paris, France, doing the 2nd year MSc in the advanced specialisation ”System Engineering: Robotics and Embedded Systems”, e-mail: lefort@ensta.fr.

Nicolas VENTROUX is with the Division of Architecture IC Design, Embedded Software at Embedded Computing Laboratory, CEA Nano-Innov, Gif-sur-Yvette, PC172, 91191 France. e-mail: nicolas.ventroux@cea.fr.

$$N_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} \quad N_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

Fig. 2: Sobel filter's masks

and minimum in the first derivative of the image. The masks presented on figure 2 are used to make all the computations, where $\mathbf{N}_x$ matrix is related to processing in the horizontal direction, and $\mathbf{N}_y$ matrix is related to the vertical direction processing.

By considering the first implementation of this method, which take into account all the elements of the matrices $\mathbf{N}_x$ and $\mathbf{N}_y$ when calculating all the values of image pixels by Sobel filtering, the time computation measured after many tests of the Sobel filter is given by 18 milliseconds, which is costly for image processing.

In Fig. 3. it is possible to observe the results achieved on the image of a patient with cerebral cancer, where the initial image is obtained by a MRI system without any filtering process. Thereafter a sequence of filtering is applied: Median filter, Threshold filter, and finally Sobel filter, finalizing the process of recognition of a tumor in a patient.

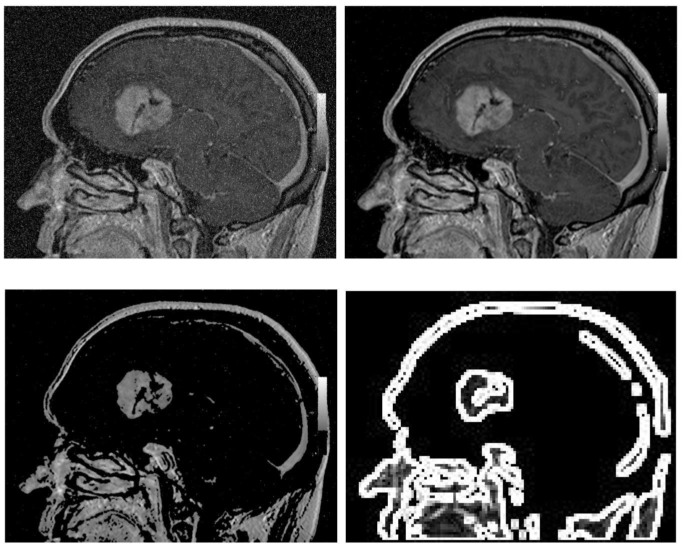


Fig. 3: First the initial image obtained by MRI process is filtered by Median filter removing the noise in the image. After that the Threshold filter is applied in order to convert grayscale or color images into high contrast. Finally the Sobel filter finishes the medical image filtering process by detecting the edge of the image.

III. Coding Optimisation

From the results of computational performance obtained in a classical implementation in C language for the three analysed filters, the importance of performing optimizations must be emphasized. In this section a coding optimisation is presented in order to highlight its first impacts on image processing time filtering.

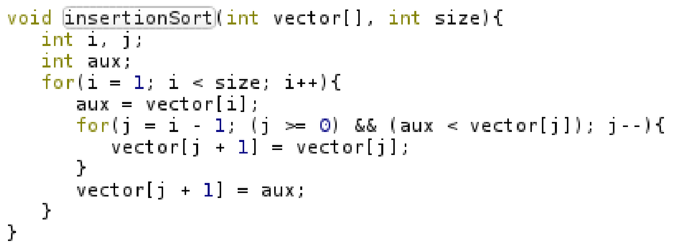
A. Local Memory Use

The default compilation of the algorithm lets the intermediate pictures on the L3 CPU cache memory which has a very high time cost access. So the first optimisation can be given by using architectures using a Generalized Shared Memory, which is maintained in a consistent state by a hardware-based coherency mechanism that operates on shared objects, wherever they happen to be calculated. In order to use this technique of memory organization the option $LOCAL\_SHARED$ is included to the main program in order to put the data on the cash memory cluster, which is formed by one or more processors, a local main memory storing both private and shared objects and an external coherency unit connected to inter-node communication means.

After include that option, the performance dramatically improves the computational and time performance of the system, from more than 400 milliseconds to approximately 40 milliseconds for the full system, comprising the process of reading and writing, and the filtering sequence taken through Median, Threshold, and Sobel filters.

B. Median Filter

The critical part of the Median filter is the sorting algorithm. Before the optimisation phase, the system used a quick sort algorithm using recursion calls. Replacing it by an insertion sort algorithm the filtering process has improved the performances by 20% .



```
void insertionSort(int vector[], int size){
   int i, j;
   int aux;
   for(i = 1; i < size; i++){
      aux = vector[i];
      for(j = i - 1; (j >= 0) && (aux < vector[j]); j--){
         vector[j + 1] = vector[j];
      }
      vector[j + 1] = aux;
   }
}
```

Fig. 4: Insertion sort algorithm implementation

Due to the high computational efficiency of the insertion algorithm with small lists (which depends on the computational power of the computer is running on), and the high efficiency with memory once this sorting algorithm needs one extra storage location to making room for moving elements, the insertion sort algorithm is used on Median filtering process of the image, which is the first filter used to highlight the cancer tumor in an medical image, in order to reduce the calculation time required for image processing. In other words, the insertion sort algorithm is statistically more efficient than a quick sort for 9 values, and does not use recursive calls in the sorting process. After running several tests with Insertion Sort algorithm is observed that this method has a small computational cost in comparison with Quick Sort, requiring a smaller processing time of about 16 milliseconds on image filtering process.

C. *Threshold*

In order to optimise the Threshold filtering process, the loop unrolling technique, which is easily applied to sequential array processing loops where the number of interactions is known prior to execution of the loop, is used in order to improve the gains by reduction the computation of complex instructions. In other words, this technique is applied by replicating the code inside the two nested loops body 27 times (which is the loop unrolling factor), decreasing the number of interactions, and yields to increase the size of the code.

D. *Sobel Filter*

The Sobel Filter can be optimized by avoiding unnecessary computations, which depend of the values of $\mathbf{N}_x$ and $\mathbf{N}_y$ matrices. In the analyzed case in this paper, by considering the matrices in Fig. 2, the following considerations can be observed:

- The central column of N_x is empty.
- The central line of N_y is empty.
- Up left and bottom right values can be reused for each mask.
- Up right and bottom left values can be reused after an reversing them.

These optimisations reduce the time computation specifically for the Sobel filter to $1,93$ milliseconds, which represents a gain of approximately 89,4% compared with the first implemented algorithm.

IV. Process Parallelization

Parallel processing divides a large task into many smaller tasks and executes them concurrently on several nodes in order to computes larger tasks more quickly. In this section a process parallelization of the code is presented by using important computational tools.

A. *Open MP*

Open MP is a standard API that can be used in shared memory programs, which are written in C and C++. It consists of compiler directives, functions, and environment variables. In the presented paper a GNU compiler is used to create the executable code, nevertheless Open MP is supported by many other compilers, and data decomposition is handled automatically. In order to run efficiently the code written for filtering image processing, just a shared memory multiprocessor architecture is used. Thus a cluster with 8 cores is used in order to divide the image in 8 parts resulting in a faster computation of the final values of each pixel in the image.

B. *Performance Analysis*

Image processing is a very relevant field for using a high level of parallelization. Thanks to its eight cores, the Virtual-SoC cluster could theoretically reduce by a factor of eight the computation time. The system was strongly optimized on the threshold and mainly the Sobel Filter. The median filter still resists. However, it should be remembered that the input picture is still on the L3 memory, explaining the lack of efficiency of the parallelization. As a last resort, we suggest to use an hardware acceleration for the median filter to beat the time target.

Filter	Without OpenMP	With OpenMP	Factor
Median	18 545 678 ns	10 966 160 ns	1,69
Threshold	240 480 ns	57 682 ns	4,17
Sobel	1 928 154 ns	437 472 ns	4,41
Total	20 714 312 ns	11 461 314 ns	1,81

V. Hardware Acceleration

Motivated by the knowledge that hardware acceleration is a technique in which a computers hardware is forced to perform faster than the standard computing architecture of a normal central processing unit (CPU), the strategy implemented as well as its description are presented in this section, in order to obtain the minimal computational time cost in filtering process.

A. *Communication with the accelerator*

In order to use efficiently the accelerator, it is important to know how to communicate with it from the memory or the processor. The process needs to send 8-bits pixels to the accelerator and the word are 32-bits. So, each word will regroup 4 pixels, written and read on the word with shifting operators and masks. Results are shown on Fig. 4. Results point out how much it is important to process the operation from the local shared memory.

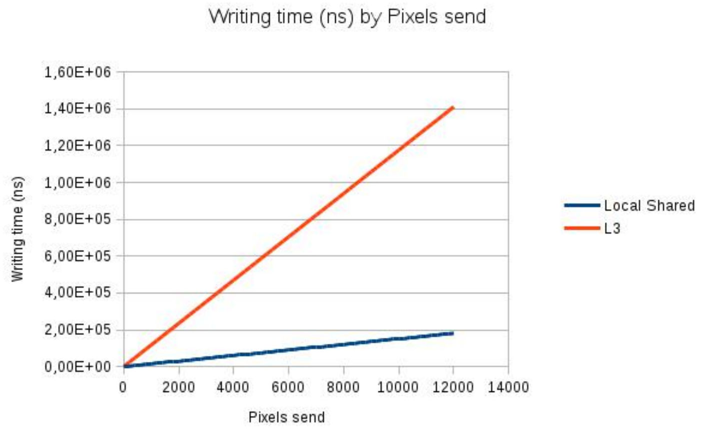


Fig. 5: Writing time on the accelerator form the processor, when the pixels come from the L3 or the Local Shared memory. Packaging process is included.

B. Acceleration strategy

Hardware acceleration of sorting algorithms is a complex issue with an abundant literature. However, the used system with a median filter using the eight neighbours of each pixel could only deals with nine values. More specifically, it only looks for the median value of them and does not have to complete the sort of the array. Moreover, hardware acceleration offers the opportunity to parallelize the process and making simultaneously several comparisons.

To make the system more reliable, it is possible to only use a very stable sort algorithm. Actually it is even possible to do the process without sorting, by using a test that assess if it is the median of a vector or not. The tested value is compared to every others and the results of each comparisons are preserved. If the difference between the number of higher and smaller value is smaller than the number of values equal to the tested one, it is a median:

$if \|bigger - smaller\| <= equal- > is\_median$

C. Hardware acceleration steps

The next section presents the different steps of the hardware accelerator based on this strategy. All of them can be pipe-lined:

- **Image buffering:** The median filter needs to work on a 3x3 area. So we will use a buffer of pixels to have a direct access to these 9 pixels at each pixel reception. Unfortunately, it increases the time process of this number of pixel reading. However it is a small value.
- **3x3 pixels loading:** This module receives the nine pixels used for computation. It is a loop shifter.
- **Comparators:** This module includes eight sub-modules which are 8 bits comparator with two outputs : one is a 2-bits value giving 1 if the first pixel is bigger, 0 if it is equal, -1 if it is smaller. The second output is one if the two values are equals, zero if none.
- **Adders:** This module contains two 8 x 4 bits registers with an output giving the sum of all their respective values. The first register receive the first output of the comparators, the second one the second output.
- **Results:** This module receives the two results of former adders. If the absolute value of the equal adder result is greater than or equal to the compare adder result, the process found the median value. The module sends it as an output. Moreover, it sends a signal to the shifter to download new values from the buffer, to start the process for the next pixel.

D. Implementation

Due to several issues from event management. Only a non pipe-lined version of the accelerator was done. It means that the shifter will wait for the acknowledgement of the result module. It will load a new set of 3x3 pixels if the result say it is an output pixel, or it will shift if it is not. Also due to event management problems, the picture will be completely loaded on the accelerator memory before the process.

The implementation was done with five threads :

- **Do shifter:** Process the shifter and initialize the process.
- **Do_compare:** Take the current values from the shifter and process the evaluation. The first pixel is compared to the 8 others. Results are written in two vectors, adder and equal. First tells if the pixel is bigger (1) or smaller (-1) and the second tells if the pixel is equal (1) or not(0).
- **Do_adder:** Add all the elements from the adder vector and return its absolute part.
- **Do_equal:** Add all the elements from the equal vector and return the result. It is processed at the sime time as Do_adder.
- **Do_result:** It compares the results of Do_adder and Do_equals. If Do_equals result is bigger, the process found a new pixel and asks the shifter to load new values.

E. Expected Results

It is possible to estimate theoretically, the time used for the new hardware median filter. The buffer can send a pixel at each clock tick. It can be assumed that the comparison process, the adding and the final step also last one clock tick each other. The worst case appears when the only median values is downloaded on the second pixel location. In this case, the loop takes 15 clock ticks : four for one process, plus eight process pipe-lined and three ticks for the return of the pixel flag. It equals to 30 nanoseconds which is more than the reading process of a pixel from the PIC. In this case, the median filter lasts $0,36$ milliseconds which is less than the time target, assuming the Sobel filter lasts $0,44$ milliseconds and the threshold $0,05$ with parallelized process. However, the statistical average of points gives a number of reduce the number of loops by a half.

F. Experienced Results

Finally after the implementation of the algorithms performance improvement, the results are measured and shown in the following table:

Process	time (ns)
Writing on accelerator	183 454
Accelerator Process	975 996
Reading on accelerator	194 906
Complete process	2 294 240
Only software (Local shared input picture)	4 111 428

The time target is not reached, but a good improvement was done. Pipelining Writing, Acceleration Process and Reading could reduce the time of the median filter under the millisecond. Moreover, the process of the accelerator can be improved by a more efficient event management.

VI. Conclusion

The implementation of the Median, Threshold and Sobel filters normally has a high computational cost, resulting in high time filtering of images. In order to avoid that problem a software/hardware optimisation of the filtering process is used, divided in two parts. In the first part a software optimisation

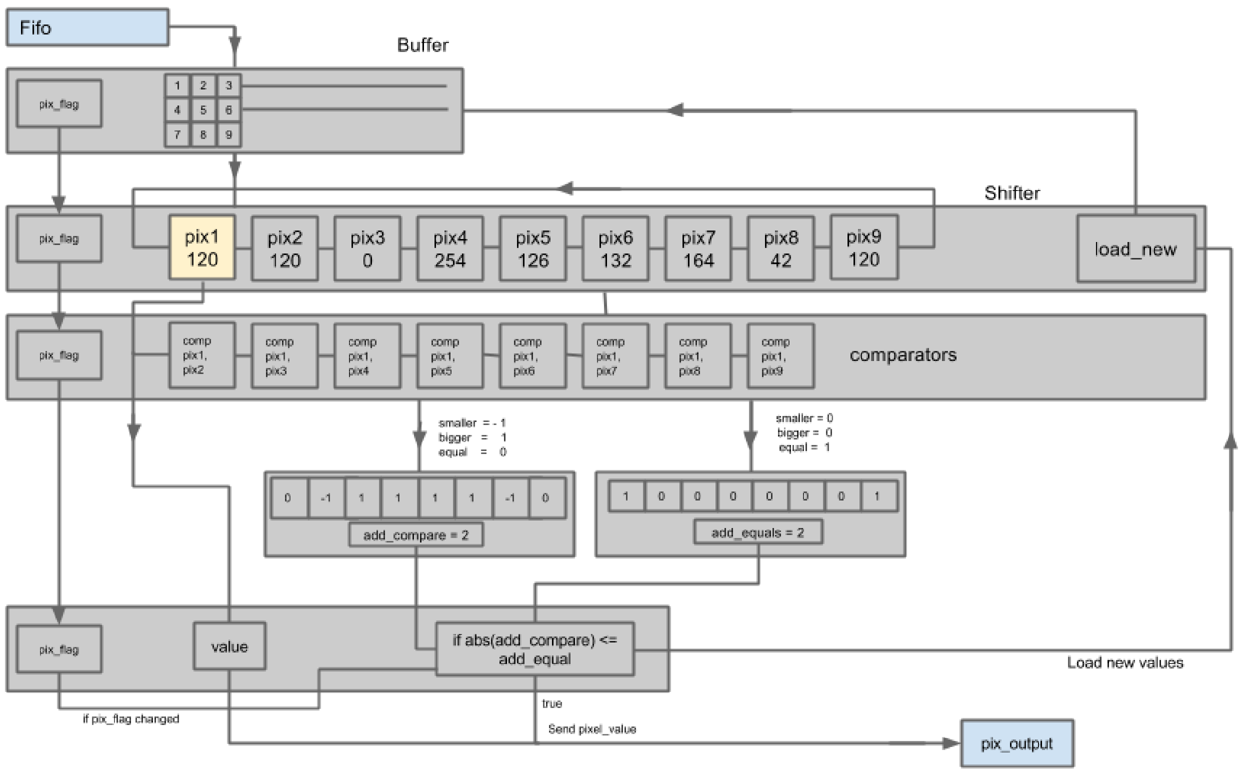


Fig. 6: Diagram of the hardware accelerator. The fifo above the diagram receives four pixels at the same time from the PIC, but it sends them only one by one. It is the opposite for the output.

using Open MP and loop unrolling are made, improving the computational efficiency of Threshold and Sobel filters. In the second part the Chip Modelling (SystemC) is used in order to decrease the computational cost of Median filter, the most costly algorithm of the 3 filters chosen in this paper.

If the accelerator helped to improve the global performance of the algorithm, the time target is still not reached. However, some possible improvement are described in this paper but not implemented or tested.

The participation of the authors in this paper, which was proposed within the context of the course SystemC at ENSTA Paristech, was at times divided into different parts between them, and at other times has made in team. Henrique VICENTE SOUZA is responsible for the first implementation process, comprising the implementation of the three algorithms (Median, Threshold and Sobel filters) in C language, which were extremely important for the initiation of the following steps, being the basis of all parts of the project. In addition He is responsible for the coding optimisation, concerning the implementation of the insertion sort in Median filter, and the loop unrolling technique in Threshold filter. After that, He did the implementation of Open MP tool, including the compiler directives, functions and environment variables in the main program. By the other hand, Alexandre LEFORT is responsible for the acceleration strategy, aiming to implement a hardware accelerator as described in Fig. 6., comprising the acceleration strategy and the hardware acceleration process.

Last but not least, regarding the work made in group, it is important to highlight that throughout the process of verification and validation of all the different stages of the project there is a participation of both authors Henrique and Alexandre, as well as in the process of local memory use, the code optimisation of Sobel filter (by avoiding unnecessary computations), and the performance analysis of all the obtained results.

Acknowledgment

This work is the result of a System on Chip Modelling (SystemC) Project at ENSTA - Paristech in Paris, France guided by Prof. Nicolas VENTROUX.

The authors would like to thank to ENSTA - Paristech for the opportunity to acquire experience in research and development, and their Prof. Nicolas VENTROUX for the guidance, all support during all steps of the project and relevant discussions that helped the authors in the process of understanding, analysing and correcting various problems of the project.

References

[1] N. VENTROUX, *Modélisation Système sur Puce (SystemC)*, Polycopie ENSTA-Paristech, Paris, France, 2014.