DISTRIBUTED ARCHITECTURE FOR REAL TIME JPEG2000 MEDICAL IMAGES DECOMPRESSION AND VISUALIZATION

J. L. García de Arboleya, R. González, P. Cosías, G. Bueno *
* Universidad de Castilla-La Mancha, E.T.S.I. Industriales, Ciudad Real, Spain

gloria.bueno@uclm.es

Abstract: New medical equipment enable integrate digital imaging technologies suitable for medical diagnosis. This paper present some of the work carried on in this field of Biomedical Informatics for jpeg200 medical images decompression and visualization. The final purpose of this work is to develop an imaging system to assist pathologist examining histological and cytological slides samples and improve the diagnosis within pathology.

Introduction

The way biomedical research deals with data at different information levels is still a challenge, [1]. Furthermore when this data is medical image, aiming to process and obtain the relevant information, [2].

This paper addresses the processing of JPEG2000 medical image for pathology. The size of pathological images is one of the major problems when analysing them. A solution to this problem has been given with the use of JPEG2000 format. This format allows progressive and partial decompression at different resolutions, which is one of advantages when reducing computational cost for display and processing, but it complicates the implementation of this visualization, [3].

In this work, we have developed a interactive image viewer, specific for pathology purposes, with the possibility of connection to an external system for fast decompression. We have also developed such a system for JPEG2000 decompression which consist on a set of programs capable to distribute the load among all the available resources.

In this introduction we will give a background over the JPEG2000 compression.

The JPEG2000 file format

The JPEG2000 compression splits the image in tiles, usually 1024x1024 pixels, converts the three colour channels to the YCrCb space and processes independently each of the channels as shown in Figure 1 and explained in [8].

The wavelet transform leaves the image in the wavelet space in squares low and high pass (LL,LH,HL,HH). Each one of the squares is called a sub-band.

The quantization step removes the frequencies that have a contribution close to zero and group the sub-bands in blocks called precincts.

The EBCOT algorithm reorganize the information in bit-planes and group them again, giving code-blocks, [4].

Finally, there is a last step of clustering, that groups all the code-blocks in layers, according to an uniform visual perception of the quality.

After compressing each of the three channels, they are compressed a second time together with the EZW [8] algorithm, and finally joined to other tiles to form a code-stream [4,8] called JP code-stream or JPC.

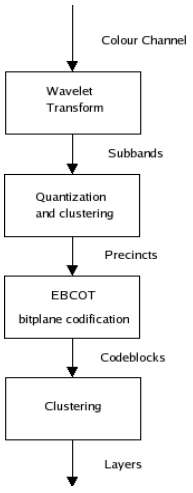


Figure 1: Compression of JPEG2000 colour channel

The JP code-stream is finally embedded in the JP2 file, which contains a header, a fine color tuning table and some other image data Figure 2.

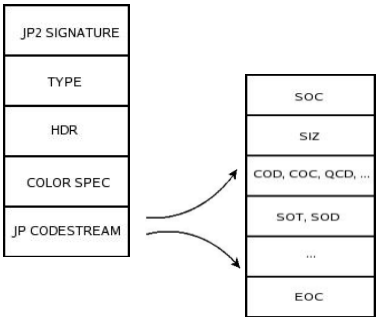


Figure 2: Internal blocks structure of a JP2 file

ISSN: 1727-1983

The EBCOT algorithm organizes the bits in such a way that optimizes the decompression later, and the clustering will divide the data into layers, to allow partial decompression [6].

Once the JPC is completed it is integrated into the JP2 file (see figure 2).

The JPEG2000 process is performed for every codified tile (a codified tile is a SOD marker segment in the embedded JPC). The required steps are [3]:

- Entropy decoder
- Region of Interest (ROI) decoder
- Decuantizer
- DW inverse transform
- Inverse intercomponent transform

The YCrCb Colour Space

This colour space was introduced as an scaled version of the previous YUV and YIQ (PAL and NCTS colour systems for TV), adapted to the luminance of the computer screens [9].

The change from RGB to YCrCb is linear and is given by the following matrix:

$$\begin{pmatrix} Y \\ Cr \\ Cb \end{pmatrix} = \begin{pmatrix} 0.2999 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \quad (1)$$

Nevertheless the image can also be stored in the HSV colour space for the processing algorithms, as it is the case for the pathology images [4].

The wavelet space

The colour components will be stored in the wavelet space. The wavelet transform is a bi-dimensional discrete one and it is applied in a multi-resolution scheme [3,5].

The wavelet transform is implemented with a progressive filter through bi-orthonormal wavelets. with scaling function [8,11]. This is explained in the following points.

The discrete wavelet transform

Usually we define continous wavelet transform as the decomposition coefficients of a function *f(t)* in the wavelet basis:

$$g(s,t) = \int f(t).y_{st}(t).dt \quad (2)$$

the discrete form of this expresion is:

$$g_{st} = \sum_k f(k).y_{st}(k) \quad (3)$$

The scaling function and biorthonormality

When we do not have a orthonormal basis in a space we still can use the dual basis of the given one to operate with vector components [8].

We define dual basis as some vectors ê such as:

$$\left\langle \hat{e}_i, e_j \right\rangle = d_{ij} \quad (4)$$

We will use this bi-orthonormal structure for performing the wavelet transform step by step, performing a high-pass filter with the mother wavelet and a low-pass filter with the dual basis, also called scaling function [8,11].

With this dual basis, we can compute the components of a vector using the scalar product as:

$$\vec{v} = \sum_i v_i.e_i = \sum_i \langle \vec{v}, \hat{e}_i \rangle.e_i \quad (5)$$

Therefore its space is orthonormal to the mother wavelet.

$$f(t) = \sum_{j,k} g(j,k).y_{jk}(t) \quad (6)$$

This equation define the scaling function from the mother wavelet where the γ functions are the basis on which we want to project our function, and the coefficients gamma are components function *f(t)* that we have decomposed.

This identity has to be true for any set of coefficients gamma coming from any possible *f(t)*.

Iterative wavelet transform schema

The low pass filter covers the frequencies not covered by the high pass filter. We can consider its output in the image space and the one provided by the high pass filter in the wavelet space.

For obtaining the complete wavelet transform we have to use all the family wavelets over the image, but this schema allow us to do it step by step.

Being the wavelet space such as its members are the mother dilations (equivalent to use the mother wavelet over the scaled image version), we can perform a second step over the low pass image with the mother wavelet again.

The problem with this schema is that the spectrum is non-uniformly covered. The low frequencies are badly covered and with every wavelet we include we reduce the interval only to the half.

Therefore the number of wavelets for a complete basis would be infinite and we will have to leave a part of the image without transforming.

We call bi-orthonormal basis to the set of a finite number of wavelets and a scaling function orthogonal to all of them, which will perform the role of the dual basis previously said.

Bi-orthonormal basis will assure that the decomposition of a signal is unique, and that the reconstruction will be possible.

ISSN: 1727-1983

The discrete wavelet filter CDF 9/7

We define CDF filter (Cohen, Daubechies, Faubeau) as the biorthonormal wavelet filter with 4 zeros direct and 4 inverse, taken with 9 and 7 taps different to zero.

The coefficients are the ones shown at the [table 1] taken from [8]

k	Low Pass	High Pass
0	0.60294901823	1.115087052456
+/-1	0.26686411844	-0.591271763114
+/-2	-0.0782232665	-0.057543526228
+/-3	-0.0168641184	0.091271763114
+/-4	0.02674874108	0.0

Table 1: Mother wavelet coefficients and the scaling function

Their graphics are plotted in the following Figure 3.

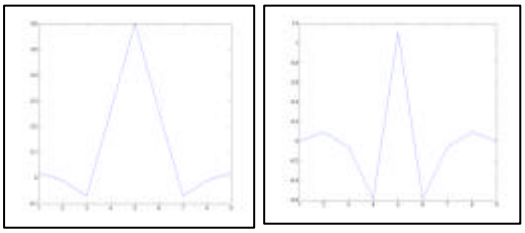


Figure 3: Mother wavelet and scaling function

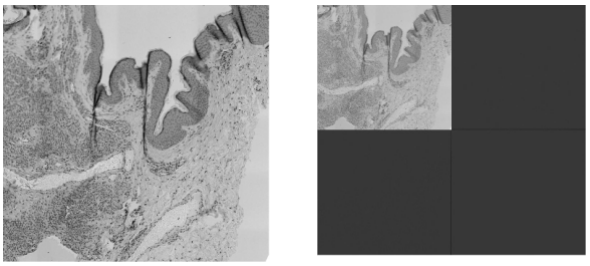


Figure 4: 1st DWT step for a histological image

The figure 4 shows the result of performing the bi-dimensional image transform, defined as the horizontal and vertical transforms of the rows and columns.

EBCOT and clustering with EZW

The JPEG2000 format is based in EBCOT (Embedded Block Coding with Optimized Truncation). This allow us to read blocks of the image secuentially, which gives a better performance thanks to the hard disk buffers in the controllers [4]

The idea of the Embedded zero tree wavelet is to store the wavelet coefficients as if it were a tree, following the path used while computing them [10]

MQ arithmetic compression

The first step, the entropy decoding, in fact are two different steps, a MQ aritmethic decoder and a EZW decoder.

The aritmethic compression is similar to the Huffmann code. In fact is a generalization of his algorithm.

Materials and Methods

The external grid

The implemented architecture is a set of servers intended to run on a multiprocessor system or a cluster. The task distribution is based on the data available and therefore no load of synchronization or interdependency for the servers is imposed. For this reason a cluster is more suitable for this purpose. This cluster has been implemented in this project Figure 5.

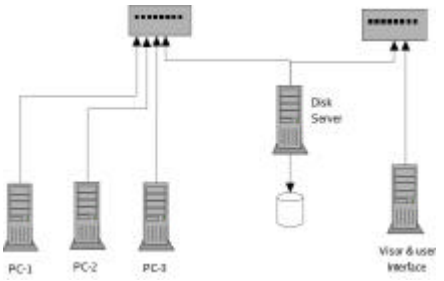


Figure 5: Network architecture of the grid

It consists of three PC computers, with processors AMD 3.2 Mhz. with a star connection, using a 100 Mbit ethernet switch, as shown in Figure 5.

This cluster is used for decompression of the JPEG2000 format but is expected to work to improve browsing speed in the client and for image processing.

For each of these purposes the cluster will run a set of specific servers that intercommunicate with each other.

Besides, though the physical connection of the cluster nodes will be a bus, each one of the servers set will consider a virtual architecture, based on the expected work loads.

The JPEG2000 decompression architecture

The decompression architecture is a three-tier structure, in which there is a server that plays the role of scheduler, other that plays the role of the block servers in the compressed file, and there is a whole middle layer of N computers that performs the decompression, as shown in Figure 6, where N=3 in this work.

ISSN: 1727-1983

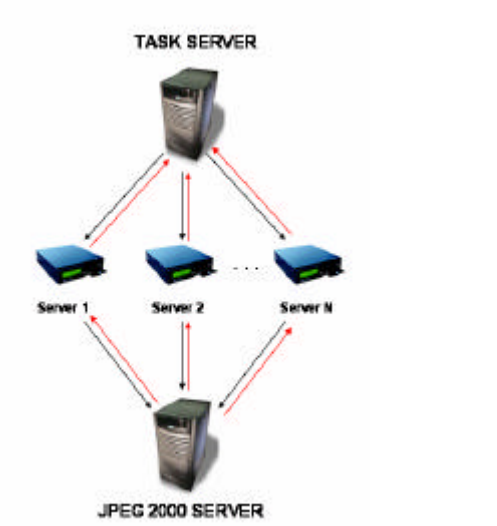


Figure 6: Server's architecture for JPEG2000 decompression

Additional tools for the cluster

We have compiled our programs with the gcc compiler. Appart of that we have installed:

- NFS, exporting the 3rd layer disk
- NFS clients in the middle layer, mounting it
- rsh, rlogind, rexec servers in the middle layer.

For monitoring the cluster load we have used the following tools:

- Network monitor: Ethereal
- CPU, memory and swap: Gnome system monitor

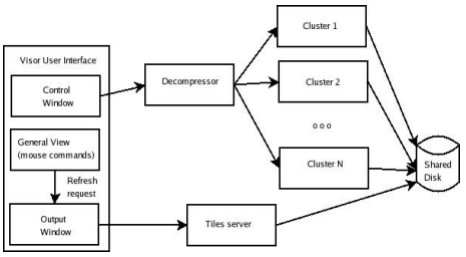
The grid at Figure 8 is implemented in general with N PC computers and a switch, being N = 4 in this case. The blocks server was implemented exporting a hard diks partition using the NFS protocol.

The main PC was a Dell Precision 370, with 3Ghz. that was running several services Figure 7:

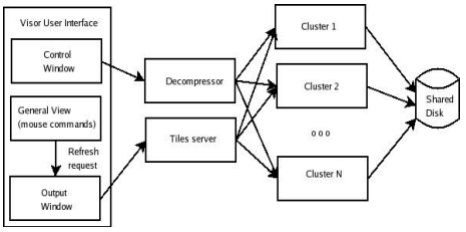
- The user control interface
- The interactive viewer for the user
- The NFS server for exporting a partition
- The shared partition with the file to decompress
- A X-server for monitoring the cluster.
- Main thread of the decompression program, or scheduler
- A server that will act as interface between the viewer and the decompressed image on disk

We have used just three nodes in out middle layer, which are three PCs with AMD 3.2 Ghz processors and 1Gb of RAM each.

Two different configurations Figure 9 were tried to compare performance. The second trial consisted in using the cluster as an intermediate cache. The direct access to the shared disk yielded a better performance, though probably this would be different with more than three nodes in the middle layer or with a disk not local. We have used the operating system Redhat Fedora 4.0 in the three nodes at the middle layer, with the NFS locking system for controlling concurrent accesses



a) Direct access from the server to the disk



b) Access to the disk using the cluster as cache

Figure 7: Two different connections from the visualization station to the external decompression cluster

The Tasks Server or scheduler

The tasks server's work is to collect requests from the user interface and to split it out into subtasks and, at the same time, to keep an actualized list of the medium layer servers and their status. We have launched two threads to perform these two tasks at the same time.

The thread for the list maintenance is simply listening in a socket and forks (creates dinamically a new process) when a connection request comes, using the standard sockets library over TCP/IP.

This new forked process will act as a handler for the connection request and will live until the connection ends. This process is responsible for registering the server's IP that it handles and for opening unix pipe for communication with it.

The other thread will wait for user interface requests and will split them among the available servers. For this purpose, the server list is shared between the two threads and standard concurrency handling mechanisms were implemented.

The communication from this second thread and the processes, which are descendents from the other thread will be done through the standard unix pipe that the first thread opens. These pipes will be alive while the handler is, and are registered in the list of available servers.

Finally, the communications between the task server and the processing servers will be done using the

ISSN: 1727-1983

standard sockets library, in a port that the server will decide (currently 6767)

The intermediate Servers

The intermediate computers run just a normal Unix daemon wich listen to requests of image parts from the task server, get the compressed image part, decompresses it and returns it to the Task Server.

Currently this servers are just working as a middle layer for a fast jp2 decompresion. Once the servers have decompressed the tile they are iddle and ready to be used in the image processing task.

In the future this middle servers will accept several commands for performing algorithms over the image. That is the overall objective of this project

The developed JPEG2000 decoder is based on the Jasper codec by Adams M. D. [3]. It has been modified as following:

- 64bit addressing for each pixel of the tile was added
- Seek instructions while writing the tiles onto disk, have been removed

Thus, we have been able to decompress an image with 25 tiles of 4096x4096 in 25 minutes in one AMD sempron 3Ghz.

The blocks servers

We have distributed the compressed image to the cluster just by sharing a hard drive by NFS and mounting it in the cluster nodes. The NFS protocol hides the request to our program.

This server is the one that contains the JP2 file. The server funcion is to distribute JPEG2000 blocks when requested. As we said before, the JPEG2000 decompresion is slow in layer 2 Figure 8 and therefore a single server from layer 3 will serve several others requests from layer 2.

The JPEG2000 decoder

This decoder is running in layer 2. The decoder developed here is only able to decompress the lossy compresion that is the one generated by the scanners we have tried.

The decoding process is similar to the one described in the introduction.

Is worthy to mention that we do not treat lossless compression in our program. For the wavelet transform we use the CDF 7/9 six times.

From the point of view of the implementation, the clients have to pass a number of parameters as strings to the server when they perform the request. These parameters codify the height and width of the window we want to obtain, the scale in the x and y coordinates and the starting coordinates of the requested window.

Results

Our aim is to develop an environment for allowing the doctors to handle JPEG2000 medical images. The load time of the image, is mainly the time it takes to decompress it. With a Sempron 3000 it took 25min. to decompress a 17000x17000 image Figure 8.

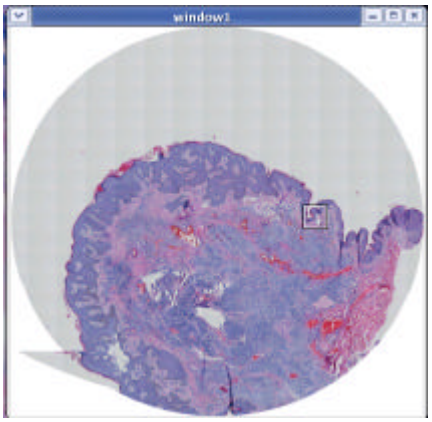


Figure 8: Image 17000x17000 used during the experiments

On a grid with three intermediate nodes we have reduced the 25 minutes of loading to 10. Adding an additional Pentium 4 to 3.2 Ghz we reduced it further to 4 min.

We have also included a button in the user interface for browsing and loading JP2 images on the local disk of the visualization station, which should be shared by NFS before sending it through the network, saving in this way bandwidth.

Load on the network while distribution the files

The load in the network and the shared drive work was in the beginning even lower than expected. This was due to the fact that while reading the tiles, we have to decompress them, being slower the process of reading that the network

The control over the cluster is performed through the scheduler Figure 9.

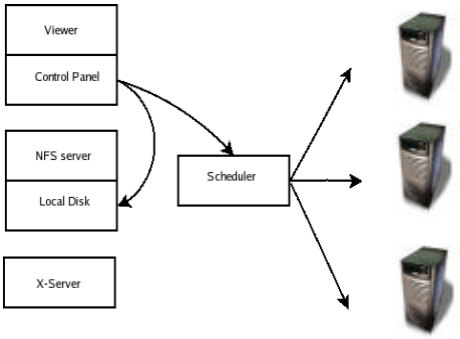


Figure 9: Visualization station and interface with the cluster

ISSN: 1727-1983

The CPU load of the three node's cluster while decompression is shown in Figure 10.

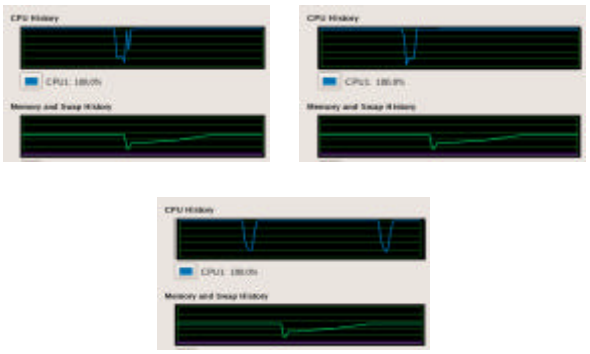


Figure 10: Gnome-system-manager running in 3 node's grid through X

The network loads while decompression is shown in Figure 11. Only the scheduler and one of the nodes are shown.

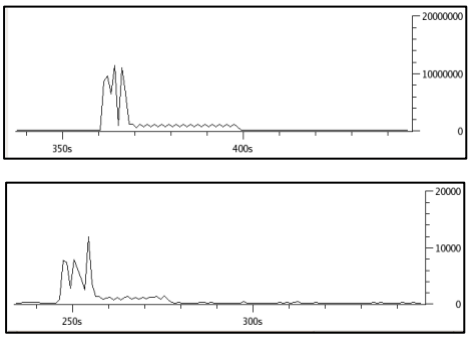


Figure 11: Network load for the monitoring machine and one of the nodes during decompression

The user interface

We want a double window. One for control user and the other for whole image display Figure 12. The user may move the frame in real time across the image, having at the same time a high resolution and global view in the 2nd window.

The problem was that having a huge image non fixable in memory and a slow network we could not give an interactive view. We have solved this by creating a buffer of adjustable size and giving a preview of this buffer in the second window while moving the mouse.

We have increased the speed rejecting mouse events produced by mouse dragging. The viewer was made on Linux, using the graphical tool Glade and the widgets library GTK+. Apart of the possibility of connection to the external decompression cluster, it has its own algorithms for independent parsing for the formats PPM, JP2 and non-compressed TIFF.

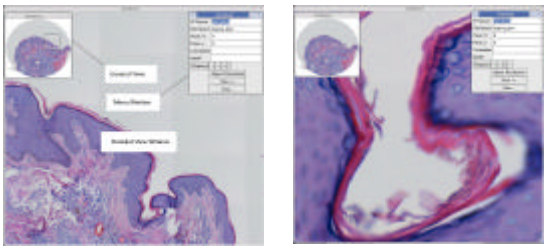


Figure 12: The viewer showing two portions of an image at different resolutions

The viewer was integrated with the command window, for browsing and opening the JPEG2000 files available in the cluster. Figure 13.

Figure 13: Control window of the viewer, able to sending decomp ression requests to the cluster

The Web interface

We have also developed a web interface to the cluster that allows the user to browse the JPEG2000 files, to order their decompressions and to visualize them Figure 14.

This interface interacts with the cluster for two purposes: Request for decompression and request for data.

It was implemented using a JavaScript embedded in the html that performs requests to a remote CGI bridge. This bridge in turn interacts with the cluster for retrieving the requested image via sockets in the port 6767.

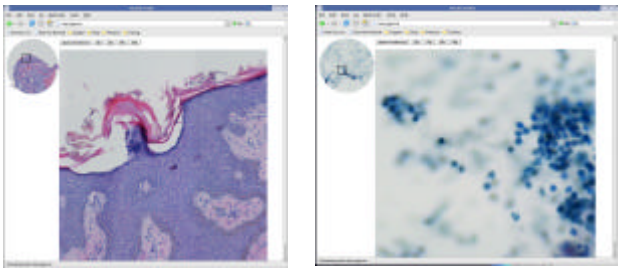


Figure 14: Firefox web browser displaying two images

ISSN: 1727-1983

Future developments

This platform will be modified tailored to the medical image processing needs. It is worthy to mention that the size of the medical images requires a parallelization of the image processing algorithms.

Usage of the cluster for RAID-like storage

At storage time, we break the JP2 file in its blocks and we send each one to one of the nodes. In this way, every node have in its local disk the tiles that

We have used the cluster for storage in such a way that the fall of a node will prevent the availability of a portion of the image.

We can make our cluster resistant to a single error (1-fall safe) if we store each of the blocks in at least two nodes.

The idea is to implement a redundant array of inexpensive disks, (RAID) using the cluster. The software for detecting the fall and reconnection of nodes will have to be implemented first.

MPI support

All the parallelization was performed with shared files and NFS locks. In the future MPI support through LAM will be included for allowing parallel image processing.

The MPI call for decompression simply allocates memory (local to each node) and fills it while decompressing the data from the NFS drive.

When needed, another MPI call will read the remote structure as required. We have implemented also MPI calls which escalate the image as required

Progressive decompression

The system described here is based in the Jasper codec. This does not allow progressive decompression.

Progressive decompression may speed up the process of previewing the image. For this reason we plan to implement it in a near future.

Acknowledgement

This research has been funded thanks to the projects INBIOMED ISCIII-G03/160 and MEC/PBI-03-017.

Conclusions

The cluster for decompression yields a processor occupation of nearly 100% due to the parallel structure of the JP2 format.

The decompression time goes down from 25:12 minutes in an AMD Sempron 3000 to 8:32 with three nodes equivalent, very close to the theoretical limit for a cluster.

The performance for visualization is not improved using a cache when the shared disk is local to the visualization station. In the case that the disk is not local to this machine, it will speed up the whole process.

References

[1] MARTÍN-SÁNCHEZ F., (2003) Synergy between Biomedical Informatics for Future Healthcare. Journal Biomedical Informatics 37: 30-42

[2] GARCIA-ROJO M, *et al*. (2001) Information System for an Anatomical Pathology Dpt., J.S. Pathological Review, 34 (2): 111-126.

[3] ADAMS, M. D. (2000) Jasper, a software based JPEG2000 codec implementation, IEEE ICIP, 117-120

[4] TAUBMAN, D. (2000) High scalable image compression with EBCOT. Proc. of IEEE. Int. Conf. of Image(3)

[5] MING G., *et al*. (2003) Computer Aided Cancer Prostate diagnosis, using image enhancement and JPEG2000. Proc. Of SPIE annual meeting

[6] JAMES C. WANG *et. al*. (2000) Multiresolution analysis of pathology images using wavelets.

[7] RIKKE D. G. GUI development with Glade2, http://www.kplug.org/glade_tutorial

[8] Lo standard JPEG2000, Instituto Politecnico di Torino, http://www.vlsilab.polito.it/Thesis/Vacca/cap1.pdf

[9] GONZÁLEZ R. C., WOODS, R. E, (1992) Digital Image Processing, Addison-Wesley

[10] SHAPIRO, J. M, (1993) Embedded image coding using zero-trees of wavelet coefficients, IEEE Transactions on Signal Processing, Vol. 41, No. 12, p. 3445-3462

[11] MALLAT, S. G. (1989) Theory for multiresolution signal decomposition: the wavelet representation; IEEE Transactions on Pattern Analysis and Machine Intelligence, 11, No. 7, 674-693.

ISSN: 1727-1983