Development Platform for Parallel Image Processing

RADU DOBRESCU, MATEI DOBRESCU, STEFAN MOCANU, SEBASTIAN TARALUNGA
Faculty of Control & Computers
POLITEHNICA University of Bucharest
Splaiul Independentei 313
ROMANIA

Abstract: - This paper describes a development distributed platform with client-server architecture that allows developing parallel primary image processing on a cluster with variable number of workstations. The principles of the software and hardware architecture of this platform are presented underlining the versatility and the capacity of adaptation to a specific application. Experimental results show that for a realistic image processing application performances are accurate and consequently the core of the architecture forms a powerful basis for automatic parallelization of a wide range of image processing software.

Key-Words: - distributed platform, scheduling, parallel processing, image segmentation and rebuilding

1 Introduction

This paper describes a development distributed platform with client-server architecture that allows developing parallel primary image processing on a cluster with variable number of workstations. In the following this platform will be named D2P3 (Distributed Development Platform for Parallel Processing). Parallel hardware architectures can be programmed using two conceptual models of parallel programming: data parallel and task parallel [1]. We have used a data (image) parallel model, in which the images are split and each part or sub-image is processed by a different processor. Previous works have proved that the application of parallelism in low level (primary) image processing can be highly beneficial. One of the most impressive syntheses was realized by a group of researchers of the University of Delft ([2], [3], [4]). They have conclude that, instead the ideal solution would be a fully automatic parallelizing compiler or at least the possibility to design a parallel programming language aimed at image processing specifically, a more practical approach is to design a software library containing parallel versions of operations commonly used in image processing. Unfortunately, there are some disadvantages in the creation of such a parallel library [5]. First, the existence of many parallel versions for different image processing procedures is very laborious. Second, the extensions are very difficult to make. Third, it will be necessary to change the source code when introducing a new platform. We feel that this solution requires too much implementation effort, is not flexible enough, and is impossible to maintain on the long term. For these reasons, we take a different approach, creating a software architecture containing a set of abstract data types and associated pixel level operations executing in data parallel fashion. The paper is organized as follows: Section 2 discusses the software architecture components. Section 3 presents the platform model as distributed system and the proposed solution for jobs scheduling. In Section 4 is described the operation mode for a typical application. The performances obtained from experiments are discussed in Section 5. Concluding remarks are given in Section 6.

2 The software architecture

The first component of the software architecture contain routines for image data partitioning, in order to indicate which data parts should be processed by each processing unit and routines for image data distribution used to scatter, gather, broadcast and redistribute data structures. The second component of the software architecture contains a large set of sequential operations typically used by image processing researchers. Each operation that maps onto the functionality as provided by a generic algorithm is implemented by instantiating the generic algorithm with the proper parameters, including the function to be applied to the individual data elements. In our current library the following set of generic algorithms has been implemented: *unary pixel operation,* in which an unary function is applied to each pixel in a given image (example: negation); *binary pixel operation,* in which a binary function is applied to each pixel in a given image (example: threshold); *geometric operation,* in which a given image's domain is transformed (example: scaling); *neighborhood operation,* in which several pixels in the neighborhood of each pixel are combined (example: median); *filtering by convolution* (example: Gauss).

Following the solution proposed Seinstra and Koelma [6] for each generic algorithm we have defined a *parallelizable pattern.* Each pattern constitutes the maximum amount of work in a sequential generic algorithm that can be performed both sequentially and in parallel (without having to communicate to obtain non-local data).

The last component of the software architecture is the scheduling component that is applied to find an optimal solution for a given application. The requests for scheduling results are performed to determine which parallelization strategy is required. The aim of *scheduling* is to provide specified shares of the total system capacity to groups of jobs.

3 Platform model and scheduling principles

Our system model consists of P processors. Processor p has capacity c_p, with $c_p > 0, p = 1, \ldots, P$. The capacity of a processor is defined as its speed relative to a reference processor with unit capacity. We assume for the general case that $c_1 \leq c_2 \leq \cdots c_P$. The *total capacity* C of the system is defined as $C = \sum_{p=1}^{P} c_p$. A system is called *homogeneous* when $c_1 = c_2 \cdot \cdot \cdot = c_P$. The platform is conceived as a distributed system.

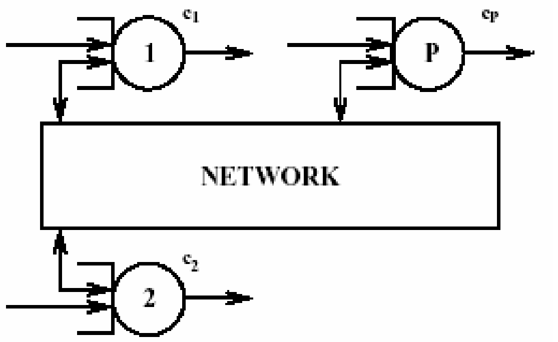


Fig.1. The model of the distributed platform

Each machine is equipped with a single processor. The main difference with multiprocessor systems is that in a distributed system, information about the system state is spread across the different processors. In many cases, migrating a job from one processor to another is very costly in terms of network bandwidth and service delay ([7], [8]), and that the reason that we have considered for the beginning only the case of data parallelism for a homogenous system. The *global* scheduling policy decides to which processor an arriving job must be sent, and when to migrate jobs. At each processor, the *local* scheduling policy decides when the processor serves which of the jobs present in its queue.

Jobs arrive at the system according to one or more interarrival-time processes. These processes determine the time between the arrivals of two consecutive jobs. The *arrival time* of job j is denoted by A_j. Once a job j is completed, it leaves the system at its *departure time* D_j. The *response time* R_j of job j is defined as $R_j = D_j - A_j$. The *service time* S_j of job j is its response time on a unit-capacity processor serving no other jobs; by definition, the response time of a job with service time s on a processor with capacity c' is s/c'. We define the *job set* $J(t)$ *at time* t as the set of jobs present in the system at time t:

$$J(t) \triangleq \{j | A_j \leq t < D_j\}$$

For each job $j \in J(t)$, we define the *remaining work* $W_j^J(t)$ *at time t* as the time it would take to serve the job to completion on a unit-capacity processor. The *service rate* $\sigma_j^J(t)$ *of job j at time t* ($A_j \leq t < D_j$) is defined as:

$$\sigma_j^J(t) \triangleq -\lim_{\tau \downarrow t} \frac{dW_j^J(\tau)}{d\tau}$$

The *obtained share* $o_j^J(t)$ *of job j at time t* ($A_j \leq t < D_j$) is defined as :

$$o_j^J(t) \triangleq \frac{\sigma_j^J(t)}{c}$$

In words, $o_j^J(t)$ is the fraction of the total system capacity used to serve job j, but only if we assume that $W_j^J(t)$ is always a piecewise-linear, continuous function of t.

Let consider as an example a system with $P = 3$, $c_1 = 2$ şi $c_2 = c_3 = 1$. For simplicity, we assume that there is no job migration and that jobs are only served by processor 1, or wait in its queue. At time $t = 0$, job 1 with service time $S_1 = 4$ enters the system, job 0 is already present, $W_0^J(0) = 2$. There are no other jobs present in the system, nor do any other jobs arrive. We consider the First-Come First-Served

(FCFS) policy [9]. Fig.2. presents $W_1^J(t)$, $\sigma_1^J(t)$ and $o_1^J(t)$.

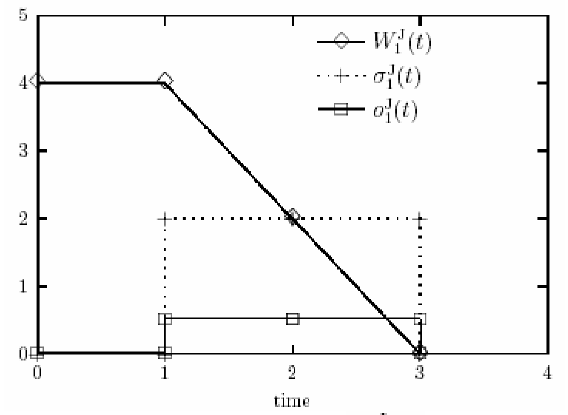


Fig. 2 The *remaining work* $W_1^J(t)$, the *service rate* $\sigma_1^J(t)$ and the *obtained share of job* $o_1^J(t)$ for the FCFS policy.

4 A case study

The main application implemented allows processing an image in 24bpp format by distributing the image blocks resulted after the segmentation realized on the Server to the associated Client applications. Fig.4 presents the ground architecture that separates the Server zone and the workstations (Client).

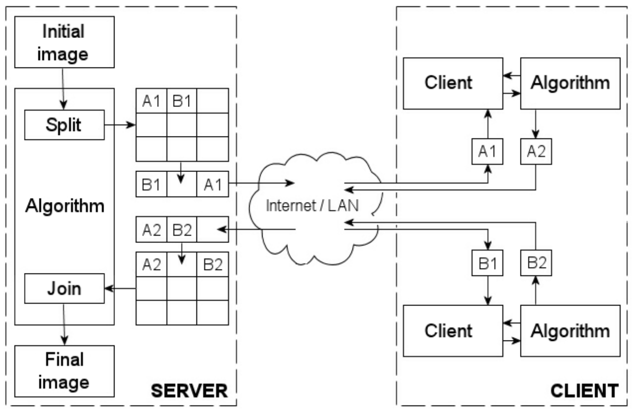


Fig. 4. The architecture of the 2D3P system

The Operation system is Windows XP and the network protocol is TCP/UDP. The current implementation contains a set of 20 algorithms, but an extension is very easy to made [10].

4.1 The description of the main application

The Server allows selecting the image to be processed together with one of the implemented algorithms with the specified parameters. Then the available free Client workstation in the local (Intranet) network are listed in order to be used to the given image processing [11]. For simplicity on the Server station was implemented a progressive segmentation method that starts with a first segment, then proceeds to areas segmentation and finally to the whole image segmentation in bitmap format. When the segmentation process is finished, the Server initiates a *Task Manager* class that supervises the processing of the resulted image blocks together with the Connection *Manager* class. When all the Image blocks are processed, the *Task Manager* transfers the control to the Server station in order to initiate the restoration of the global image. The modules of the main routine are listed in table 1.

Table 1. Description of the Application Modules

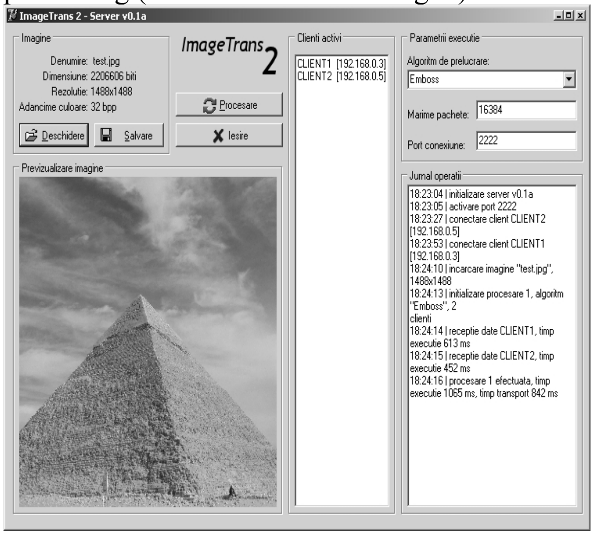
Module	Description
Server.exe	Allows the selection and the launch of the processing algorithms on the available Client stations in LAN.
Client.exe	Executes at demand the processing of an image block by applying the algorithm solicited by the Server.
Common.dll	The common library containing the code for the TCP communication, the registration of the events and the planning.
Algorithms.dll	The library containing the acquired or implemented algorithms.
Alginterface.dll	Associated library used in the Server application to introduce the specified parameters of the algorithms.
Geometry.dll	Supported algorithms: Flip • Rotate • Mirror
Neighbor.dll	Supported algorithms: EdgeSobel • GaussianBlur • MeanRemoval • Sharpen •Smooth • Shapen • Median
Pixel.dll	Supported algorithms: Brightness • Color • Contrast • GrayScale • Invert(Negative) • Emboss • Blur • Noise

4.2 The description of the communication procedures

The main application, that consist in still image segmentation, clustering, data block distribution, image processing on individual hosts and then processed blocks transfer and grouping in order to obtain a whole processed image was conceived to be tested first on a simulation model [12]. The transfer of data was simulated on RTD channels [13]. A number of experiments have been performed to test both functional and performance aspects of the service. All tests were run on a experimental platform with a cluster of until 8 Pentium2 PCs connected via a dedicated 10 Mbps Ethernet hub, but the intention is to repeat the experiments in a wireless network. In this aim some aspects concerning the effects of temporal and spatial heterogeneity of channel bandwidth [14] were taken into account. Finally, the real test were run in a typical cluster with Client-Server architecture. The Server maintains a permanent TCP connection with each of the Client stations. An accidental interruption of the connection is signalled by a data packet of zero length. In the case of the Server, *Connection Manager* is implied in the treatment of errors by releasing the system resources. In the case of a Client the detection of a error produce in addition the change of the message of the Status Bar in „*Disconnected*". The reception is asynchronous realized in a circular buffer of 4096 bytes. After the Server sends the processing algorithm, the configuration parameters, and the dimension of the next image block the Client can reserve the necessary memory space. When the processing operation is finished, the Client station sends the result to the server in a similar manner. The Server sends periodically broadcast packets in order to maintain the synchronization and to ensure the possibility of a dynamic connection or disconnection of a Client station.

5 Implementation, tests and results

The Server interface has many dialogue windows, that allow to display images at different stages of processing (see the screenshot in fig. 6).



CLIENT1 [192.168.0.3]
CLIENT2 [192.168.0.5]

Algoritm de prelucrare: Emboss
Marime pachete: 16384
Port conexiune: 2222

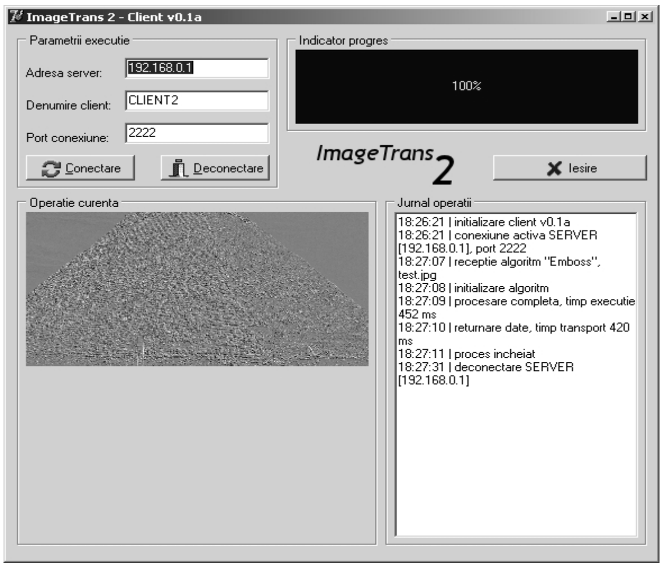
```
18:23:04 | initializare server v0.1a
18:23:05 | activare port 2222
18:23:27 | conectare client CLIENT2
[192.168.0.5]
18:23:53 | conectare client CLIENT1
[192.168.0.3]
18:24:10 | incarcare imagine "test.jpg",
1488x1488
18:24:13 | initializare procesare 1, algoritm
"Emboss", 2
clienti
18:24:14 | receptie date CLIENT1, timp
executie 613 ms
18:24:15 | receptie date CLIENT2, timp
executie 452 ms
18:24:16 | procesare 1 efectuata, timp
executie 1065 ms, timp transport 842 ms
```

Fig. 6. A screenshot of the main Server page

So, first one can see the original image and then the image after segmentation with a mark for each image block. After the end of an image block the resulting block is superimposed on the old position. This process continues until all the result image blocks are imbricate in the right position and the whole result image is displayed.

At this stage of development the Application offers a limited set of *Processing functions.* This list contains functions with a different degree of complexity, from the simplest unary operations to the most complicated convolution operations. Our aim was to determine when the parallel processing becomes efficient. For the moment, the functions included in Applications are: Flip, Mirror, Negative, GrayScale, Emboss k3, Blur k3, Gaussian k3, Gaussian k5, Soften k3, Shapen k3, Edge k3, Median k7 and Noise. The Symbol k followed by a number indicates the size of the convolution kernel. In the right side of the main Server window we have the box named *Segments size* which contains the information necessary for the image segmentation procedure.

The *Client* interface (represented by the screenshot in fig. 7) is structured in a similar manner as the Server interface. The right side represents the control zone, the left side is the graphical zone where is displayed the image block after processing. By acting on the button *Connect* Client try to connect at the specified IP and port. By acting on the button *Close* the Client is disconnected. In the left side there are placed two boxes.



Adresa server: 192.168.0.1
Denumire client: CLIENT2
Port conexiune: 2222

```
18:26:21 | initializare client v0.1a
18:26:21 | conexiune activa SERVER
[192.168.0.1], port 2222
18:27:07 | receptie algoritm "Emboss",
test.jpg
18:27:08 | initializare algoritm
18:27:09 | procesare completa, timp executie
452 ms
18:27:10 | returnare date, timp transport 420
ms
18:27:11 | proces incheiat
18:27:31 | deconectare SERVER
[192.168.0.1]
```

Fig. 7. A screenshot of a Client page, during processing

When the user ask to process many images, the Server creates a queue of tasks (a pool), and the Client verifies permanently this queue in order to see if there is something to process. At this moment the application can run in two modes. In the first mode, the user selects the image to process, the segmentation procedure and the processing algorithm [15]. In the second mode, the Server makes an optimal distribution of the load for each connected Client station in the LAN [16]. The number of block images is the same as the number of working Client stations. After a processing operation start, the Server can dynamically adjust the load by using two parameters: the processing time communicated by a Client station after running a test file and the transmission times that result from the difference between the total processing time and the processing time for an image block. Other two benchmarks are considered too: the ratio between the size of a test file and the size of the image to be processed, and the ratio between the necessary time to process the same image block with the test function and respectively with the chosen processing function.

All the tests were made only with Client stations having the same properties and the same set of specified processing functions. This option offers the possibility to determine when a parallel image processing becomes efficient. That depends essentially from the size (and consequently on the number) of the image blocks obtained by segmentation. Actually, the size of an image block allotted to a Client station is determinate by four parameters: 1) the transmission and receiving time of a block to and from the Client station, t_{send}.; 2) the necessary time to process this image block on the Client station t_{proc}; 3) the ratio between the size of the image to be processed and the size of the test image; 4) the ratio between the processing time of a Client station for the test function and for the function to be used in the processing operation.

Table 2 presents the processing times for different test functions applied on an image of 2300x1600 pixels at 24bpp and the ratio between the transmission time and the processing time for each function and Table III contains the values for the effective processing time (including the transmission and receiving times and the time for the reconstruction of the global image) obtained for the same functions in the situation when a different number of Client stations have participated to the processing, respectively 2, 4, 6 or 8 Client stations.

Table 2. Indicators for the Test Functions

Function / Code	t_{proc}	Ratio
Median k7 / Mk7	23.141	0.300
Gaussian k5 / Gk5	4.765	0.161
Gaussian k3 / Gk3	1.656	0.071
Blur k3 / Bk3	1.672	0.072
Emboss k3 / Ek3	1.422	0.061
Noise/Noi	0.781	0.033
Negative/Neg	0.078	0.003

Table 3. Indicators for the Effective Processing Functions

Code	t_{proc} - n=1	$t_{proc\ total}$ - n Client processors			
		n=2	n=4	n=6	n=8
Mk7	23.141	14.233	9.670	8.444	8.016
G k5	4.765	2.167	1.478	1.182	1.166
Gk3	1.656	1.288	0.954	0.766	0.742
Bk3	1.672	1.172	1.022	0.804	0.798
Ek3	1.422	1.065	0.920	0.756	0.756
Noi	0.781	0.533	0.496	0.322	0.346
Neg	0.078	0.065	0.078	0.086	0.098

The results confirm that the proposed solution implements an efficient algorithm for parallel color and grey level image processing using convolution functions. The experiments show that a border overload is considerable less than the overload resulted from the transmission of all the pixels of the image, especially when the size of the kernel is significantly less than the size of the image. Only in this situation (typical for convolution processes) the segmentation method is not an essential component in establishing the total processing time and the performances of the distributed processing. Another condition to obtain good results is that the Intranet works at optimal parameters (a collision rate less than 10% and a loading rate less than 35%). Using to many processors units can be counterproductive. From Table 3 one can observe that the processing time decrease almost linear with the number of processor units, when the reference processing time (with a single processor) is greater than 1 second, while when this time is less than 1 second, a number of processors units greater than 6 leads to the limitation or ever to the diminution of the performances.

6 Conclusions and future work

In this paper we have described a software architecture that allows an image processing researcher to develop parallel applications in a transparent manner, on a development platform implemented as a collaborative distributed system. In the same time this architecture offers the possibility to test several modes for tasks management and scheduling and to try an optimization of the load balancing between the workstations. Experiments show that the proposed procedures are highly accurate for parallel processing using convolution functions. In the near future we will focus our attention on the improvement of the scheduling component, by using workstations with different processing capacities and also other service policy for the queue of jobs, for example *Processor Sharing* [17]. We will continue implementing example programs to investigate the implication of parallelization of typical applications in the area of real-time image processing, trying to improve the performances by supporting the execution of a sequence of algorithms on the same block, dynamical reconstruction of the processed image and extension of the functions library.

References:

[1] C. Nicolescu and P. Jonker, A data and task parallel image processing environment. *Parallel Computing,* 28(7-8), 2002, pp.945-965

[2] D. Hammerstrom and D. Lulich. Image Processing Using One-Dimensional Processor Arrays. *Proc. of IEEE,* 84(7):1005-18 (1996)

[3] F.J. Seinstra, D. Koelma and J.M. Geusebroek. A Software Architecture for User Transparent Parallel Image Processing, *Parallel Computing,* 28 (7-8), 2002, pp. 967-993

[4] C. Soviany, Embedding Data and Task Parallelism in Image Processing Applications, *PhD Thesis*, Technische Univ. Delft, 2003

[5] T. Bräunl, *Parallel Image Processing*, Springer-Verlag, Heidelberg, 2001

[6] F. Seinstra and D. Koelma, Lazy Parallelization: A Finite State Machine Based Optimization Approach for Data Parallel Image Processing Applications, *Proceedings of the 17th Int. Parallel & Distributed Processing Symposium IPDPS 2003*, p.229-236

[7] G.Agosta, S. Crespi Reghizzi, G. Falauto and M. Sykora, JIST: Just-in-Time Scheduling Translation for Parallel Processors, *Third Int. Symposium on Parallel and Distributed Computing - ISPDC04*, 2004, pp. 122-132

[8] S. Papavassiliou, A. Puliafito, O. Tomarchio, and J. Ye, "Mobile Agent-Based Approach for Efficient Network Management and Resource Allocation: Framework and Applications", *IEEE J. Selected Areas in Comm.*, vol. 20, no. 4, 2002, pp. 858-872

[9] I. Stoica, H. Abdel-Wahab and K. Jeffay. On the Duality between Resource Reservation and Proportional Share Resource Allocation. In *Proc. Multimedia Computing and Networking*, 1997 pp. 207–214

[10]M. Dobrescu, A Client-Server Image Transfer Application for a Distributed Processing System, *U.P.B. Scientific Bulletin*, Series C- Electrical Engineering, Vol. 65, No. 1-4, 2003, pp. 77- 85

[11]X. Li, B. Veeravalli, and C.C. Ko, "Distributed Image Processing in a Network of Workstations", *International Journal of Computers and Applications*, ACTA Press, Vol. 25 (2), 2003, pp. 136-145

[12]M. Dobrescu and St. Mocanu - Resource management for real time parallel processing in a distributed system, *WSEAS Transactions on Computers*, Issue 3, vol.2, 2004, pp.732-737

[13]M. Hiltunen, R. Schlichting, X. Han, M. Cardozo and R. Das, Real-Time Dependable Channels: Customizing QoS Attributes for Distributed Systems, *IEEE Transactions On Parallel And Distributed Systems*, **10**, no. 6, 1999, pp.600-612

[14]Jun Huang and S.-Y. Lee, Effects of Temporal and Spatial Heterogeneity of Channel Bandwidth on Performance of Individual Messages in a Heterogeneous Communication Networks, *2004 International Conference on Parallel Processsing*, 2004, pp.126-133

[15]K. Shen, H. Tang, T. Yang and L. Chu, Integrated Resource Management for Cluster-based Internet Services. *Proceedings of Fifth USENIX Symposium on Operating Systems Design and Implementation (OSDI '02)*, 2002, pp 225-238

[16]C. Denis, J.-P. Boufflet, P.Breitkopf, A Load Balancing Method for a Parallel Application Based on a Domain Decomposition**,** *Proceedings of the 19th IEEE Int.l Parallel and Distributed Processing Symposium (IPDPS'05),* 2005

[17]M. Dobrescu. Distributed Image Processing Techniques for Multimedia Applications, *Ph.D. Thesis*, Politehnica Univ. of Bucharest, 2005