On Real Time Image Processing on a Network of PCs

Pere Millán and Eduard Montseny

Abstract—**Many image processing algorithms have a very high execution time if only a processor is used for processing them. Using a SIMD parallel structure for its execution could reduce this time. This is particularly important in the case of algorithms that must be processed in real time. The use of networks of PCs is an appealing solution that besides its low cost, takes advantage from both, the high speed of actual interconnection networks, and the high-performance of PCs. In this paper we present a model that explicitly considers system parameters, network parameters, and application parameters. So, the speed and communication model of the considered network, the workstations and PCs computing power, the per-pixel computational cost of the algorithms (that can be constant or variable), and a variable number of computers have been considered.**

We don't aim to evaluate the processing of high and medium-level algorithms of a MISD structure, but we present the first results of our evaluations for iterative low-level image processing applications. Specifically, we give a prediction model to distribute the data to each processor of a distributed system, minimizing the processors idle time.

Index Terms—**Distributed systems, Load balancing, Low-level image processing, real time image processing.**

I. Introduction

TYPICAL real-time computer vision tasks require huge amount of computing power, larger than can be achieved by current state-of-art workstations and personal computers. Parallel processing appears to be the only solution [1] to obtain enough computing power for handling real-time computer vision applications. As an alternative, special-purpose hardware or vector supercomputers have often been used.

In the nineties parallel processing started to dominate the supercomputers and more specifically the mesh machines. But from the end of the nineties it has increased the interest in distributed parallel computing. Although this architecture has been available for many years, two developments have served as catalysts to the rapid growth in the use of clusters-based computing. First, high performance workstations and PCs with microprocessors that challenge custom-made architectures are widely available at relatively low cost. Second, several software packages have been developed to assist the programmer in process management, inter-process communication, and program monitoring/debugging in a distributed environment [2], [3]. Moreover, the increasing performance of inter-connection networks as for security and data transfer speed has also been crucial for increasing the use and interest in the research of this kind of architecture.

A PC cluster comprises a certain number of general-purpose computers interconnected by a local area network (LAN). Besides their cost advantage over other architectures, very powerful nodes with large memories and I/O capabilities characterize clusters of PCs. Also, clusters are relatively flexible, in that additional computing power and communication capacity can be easily configured into the system.

An increasingly number of low-level image processing algorithms and a lot of medium-level ones, have SIMD structure. Among the first, are especially usual those which, after evaluating image characteristics, analyse them at a local level and just at the final of the process take a decision about classification and scene interpretation. A good solution for reducing the processing time of this type of algorithms consists in processing them within a general-purpose computers network.

However, two important problems must be considered when using this kind of platform: the way tasks have to be distributed among the network nodes and the number of nodes to be used so that the algorithm can be processed with the minimum execution time.

In this paper, we have studied several ways of partitioning an image for distributing the workload among a network of PCs that will perform a low-level processing task. The restrictions imposed to the system are as follows:

a) One of the PCs owns the image data, and act as a master.
b) All the working processors must process the same volume of data.
c) There is a fixed computational cost per pixel.
d) The results obtained for each pixel have to be sent back to the master.
e) The image is partitioned into a number of areas that depends on the number of processors, and the size of all these areas is almost the same according to b).

The rest of the paper is organized as follows. Section II

This work has been supported in part by the Spanish Ministry of Science and Technology under projects DPI2004-07993-C03-03 and TIC2002-02791.

P. Millán is with the Computer Engineering and Mathematics Department, University Rovira i Virgili, Av. Països Catalans, 26, 43007 Tarragona, Spain (e-mail: pere.millan@urv.net).

E. Montseny is with the Computer Engineering and Automatic Control Department, Technical University of Catalonia, Pau Gargallo, 5, 08028 Barcelona, Spain (phone: +34-93-4011691; fax: +34-93-4017045; e-mail: eduard.montseny@upc.edu).

introduces load-balancing restrictions. Section III summarizes load-balancing strategies. In the next two sections we will explain how to obtain the optimum size of the data messages, the number of communications needed to distribute the image and to collect the results from each one of the slave processors. We present some comparative results at section VI assuming two types of interconnection networks: Ethernet [4] and Myrinet [5]. Finally, section VII presents some conclusions and ongoing and future developments.

II. Load-Balancing restrictions

The load-balancing problem is central to all approaches of parallel processing. Many studies have been carried out on both, static and dynamic load balancing strategies [6], [7]. The workload can be either in the form of tasks or pieces of data. In the case of a network of PCs performing low-level computer vision algorithms, which uses a SIMD structure, the only solution for load balancing consists in a correct workload distribution among the different processors. This is because for such parallel applications, it is quite inefficient to migrate the processes because of the large overheads that may incur [2].

For modeling our load-balancing scheme, we have considered that:

- All the processors are only devoted to process the vision algorithm.
- All the processors hold the same computing features.
- All the processors analyse the same volume of data.
- One processor act as a master and the others (slaves) are devoted to process the data.
- We only consider algorithms that need to be executed in real time.

The workload distribution should be performed attending to the objective of minimising the idle time of the processors and, at the same time, minimising the total execution time of the algorithm.

III. Load-Balancing Strategy

The execution model of our study consists of a master processor and a collection of slave processors. The master is responsible for partitioning a given image into a set of sub-images, distribute them to all the slaves' processors and collect back the results (Fig. 1).

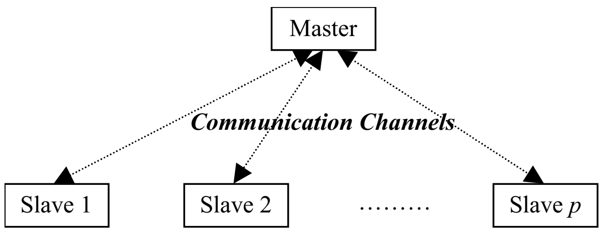

Fig. 1. Execution model.

It is supposed that the slave processors can compute and communicate in parallel. This could be achieved using, for example, MPI-2 One-sided Communications [8] at the software level, and Remote DMA [9] at the network level. So, it will be the responsibility of the master assuring that all the slave processors have data to be processed, and asking the slaves for the results in the most efficient way.

The image processing tasks considered in this work involve mainly low level image processing algorithms. For this type of image operations, a new pixel's value depends on the pixel itself and its neighbors. Therefore, after partitioning the image, each slave is expected to process a block of consecutive pixels rather than a set of unrelated pixels. In order to make the communication simple and efficient, a strip-wise partition is adopted since images are stored by rows in the memory. Moreover, it has to be taken into account that the transfer of data stored in consecutive memory positions has less communication cost [10].

As shown in Fig. 2-a, the first function of the master consists in partitioning the image into as many strip-wises as slave processors the system owns, in such a way that each strip-wise has the same number of lines (rows). Then, the master assigns each strip-wise to one of the processors, that is, the master sends the strip data by smaller messages to the slaves in the most efficient way.

Next, the master divides internally all the strip-wises into sub-blocks of data (lines) of variable size (Fig. 2-b). The master uses a message for sending each sub-block to the corresponding slave processor.

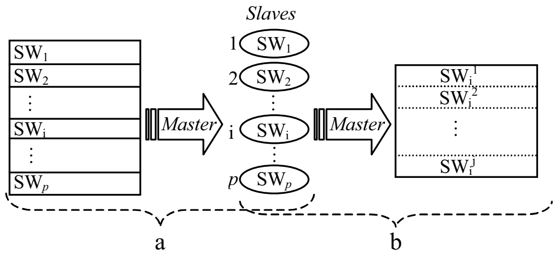

Fig. 2. Internal partition method used for sending the data to the slaves.

The master will collect, using messages, the results obtained by the slave processors. For sending the results, the slaves will group together the results into sub-blocks of variable size, so that the communications can be carried out in the more efficient way. All the slave processors will group the results in the same way, but according to the size specified by the master, that is in charge of calculate them. All these actions must be performed minimising total system execution time.

It is assumed that the first task has a fixed computational cost that will be reduced as much as possible, considering that during the time the master is busy distributing the image, the other processors are idle and waiting for the data.

Next we will study in detail the tasks of data distribution and collection of results.

IV. Data Distribution

Let us start by defining some parameters used at this and next sections.

p	number of slave processors in the system.
m	number of lines/rows of the image.
n	number of pixels (columns) in each line of the image.
SW_i	strip of rows associated to slave processor *i*.
SW_i^j	sub-block *j* of strip *i*.
RB_i^j	results obtained by processor *i* for the sub-block *j*.
t_s	start-up message communication time.
t_w	per-byte transfer time.
t_c	per-pixel processing time.
T_c^i	time taken by processor *i* to compute all its data.
t_{sBj}	time taken for sending sub-block *j* of a strip.
t_{pBj}	time taken by a slave to compute sub-block *j*.
t_{sRj}	time needed for collecting the sub-block *j* of results.
t_{pRj}	time taken by a processor for obtaining the sub-block *j* of results.
RB	number of resultant bytes per pixel.
$N(T)$	maximum number of messages that can come about.
N_{bp}	maximum number of bytes per network packet.

As our study is focused on algorithms that must be executed in real time, one of the main aims of our model is to minimize total image processing time. So, we will try to carry out in parallel the distribution of data to all the slaves, and the task of data processing by the slaves.

For verifying if this parallel processing is possible, the master must evaluate if the maximum number of transfers, including data and results, can be performed during the time the slaves are processing their own information.

This will be possible if next conditions are satisfied:

- The time taken by slave processor i to process all its own data, as given by (1), must be equal or greater than the time needed by the master for distributing all the data and collect all the results from all the slaves (2).

$$T_c^i = \left(n \cdot m \cdot t_c\right) \big/ p \qquad (1)$$

$$T_c^i \geq N(T) \cdot t_s + \left(1 + RB\right) \cdot n \cdot m \cdot t_w \qquad (2)$$

The case in which (2) is not verified, is not analysed in this work but it will be considered in future research.

Once the master has obtained all the parameters, it would have to verify that *N(T)* is greater than the real number of necessary transfers.

In order to the master can distribute all the data, first of all it must calculate the size of all the messages. For getting the size of the messages, two conditions have been taken into account. The first of these conditions can be enounced as follows:

Condition 1: *For each slave processor i, its first block of data is the smallest assuring that*:

1-1- All the processors can begin their computation with the minimum delay.

1-2. When the first processor receives the second message with data SW_i^2, it still has first message data to be processed.

So, the amount of data within the first message (SW_i^1) must contain enough lines as to assure that all the processors can begin to work with the minimum delay and guaranteeing that, when the data have been sent to the last processor, the first one still has data to continue working, as can be deduced observing Fig. 3.

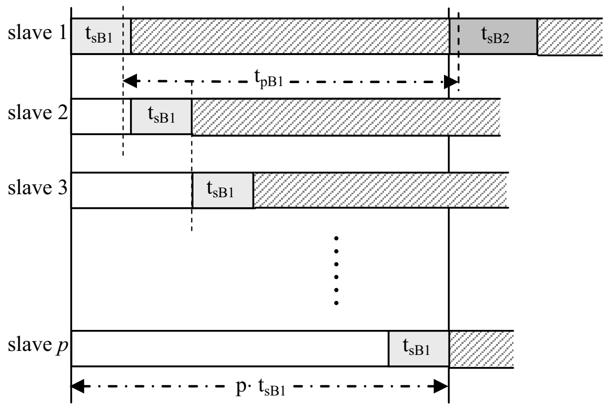

Fig. 3. First communication/processing time scheme.

If the first block of data contains *b* lines of pixels, that will be processed, plus the *a* lines that are necessary to process the *b* previous ones, if the network fragments message into packets of at most N_{bp} bytes, the communication time of all these lines will be given by:

$$t_{sB1} = \left\lceil {(b+a) \cdot n} \big/ {N_{bp}} \right\rceil \cdot t_s + (b+a) \cdot n \cdot t_w \qquad (3)$$

Otherwise, the communication time will be given by:

$$t_{sB1} = t_s + (b+a) \cdot n \cdot t_w \qquad (4)$$

Moreover, as processing time of first block is given by:

$$t_{pB1} = b \cdot n \cdot t_c \qquad (5)$$

The number of lines (*b*) will be the first integer value satisfying the following condition:

$$p \cdot t_{sB1} \leq b \cdot n \cdot t_c \qquad (6)$$

Second condition to obtain messages' size is the following:

Condition 2: *After the first message has been sent, the remainder data have to be distributed with the minimum number of messages, but taking into account that the processors must have at all times data to be processed.*

The number of messages must be minimised since the connection has to be established with each message in order to transfer the data (t_s). This implies that the new data have to be delivered to all the processors in a time equal or lower than the processing time a slave needs for processing the data received in the previous transfer.

So, while the last slave is computing the data of first message, the master besides of finishing sending first message to the last processor it must send the second message to all the slaves. It implies that the number of lines of the second message have to be equal to the maximum number verifying that computing time is greater than sending time. This situation can be observed in the second communication/processing time scheme depicted by Fig. 4.

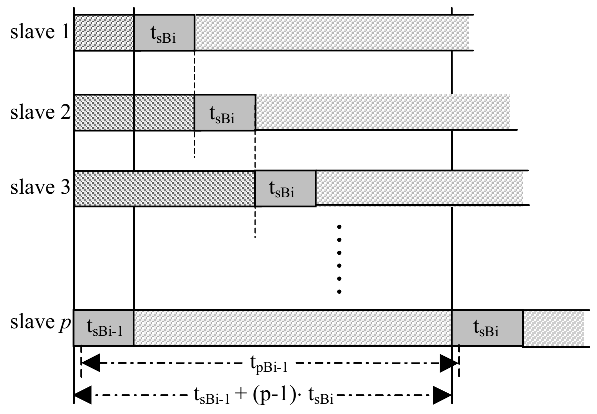

Fig. 4. Second communication/processing time scheme.

As processing time of first data block is given by (5), if we define l_2 as the number of lines of the image delivered by the second message, if the network fragments messages, the following inequality must be satisfied:

$$b\cdot n\cdot t_c \ge (p-1)\left[\left\lceil \frac{l_2\cdot n}{N_{bp}}\right\rceil\cdot t_s + l_2\cdot n\cdot t_w\right]+ \left\lceil \frac{(b+a)\cdot n}{N_{bp}}\right\rceil\cdot t_s+(b+a)\cdot n\cdot t_w \tag{7}$$

Otherwise, the inequality to be satisfied is:

$$b\cdot n\cdot t_c > (p-1)\left(t_s+l_2\cdot n\cdot t_w\right)+t_s+(b+a)\cdot n\cdot t_w \tag{8}$$

Generalising previous inequalities, if l_i is the number of image lines delivered in the i-th message, and the network fragments the messages, the inequality to be satisfied is:

$$l_{i-1}\cdot n\cdot t_c \ge (p-1)\left[\left\lceil \frac{l_i\cdot n}{N_{bp}}\right\rceil\cdot t_s + l_i\cdot n\cdot t_w\right]+ \left\lceil \frac{l_{i-1}\cdot n}{N_{bp}}\right\rceil\cdot t_s+l_{i-1}\cdot n\cdot t_w \tag{9}$$

Otherwise:

$$l_{i-1}\cdot n\cdot t_c > (p-1)\left(t_s+l_i\cdot n\cdot t_w\right)+t_s+l_{i-1}\cdot n\cdot t_w \tag{10}$$

Equations (3) to (10) allow us to evaluate the size and the number of different messages that have to be delivered to each slave processor of the system.

V. Collection of Results

The collection of results by the master must be carried out while the slaves are finishing the processing of their image data. So, the computation of the message's size of the results has been performed in reversed order with regard to the order the communication of those messages is done.

This is why, first of all, we calculate the size of the last message of results and then we deduce the size of the remaining messages. These calculi have been carried out considering that all the slaves have the same computation time, and that all of them begin their processing with some delay with regard the previous one, as depicted in Fig. 5.

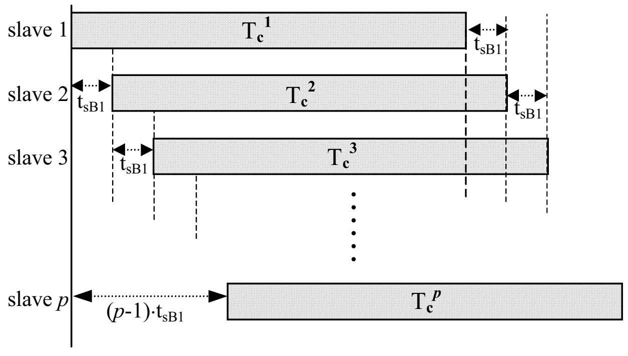

Fig. 5. Computation/initial delay time scheme.

For getting the size of the messages of results, two conditions have been taken into account. The first of these conditions can be enunciated as follows:

Condition 3: *The last message of results is the biggest assuring that:*

3-1. *All the processors can start sending the last results with the minimum delay (or immediately after they finish their computation).*

So, the number of results within the last message (Rn) of the 1st processor must contain the maximum number of lines as to assure that the 1st processor has sent the whole result message when the 2nd processor has finished its work.

A scheme of this processing/collection behaviour is depicted by next figure.

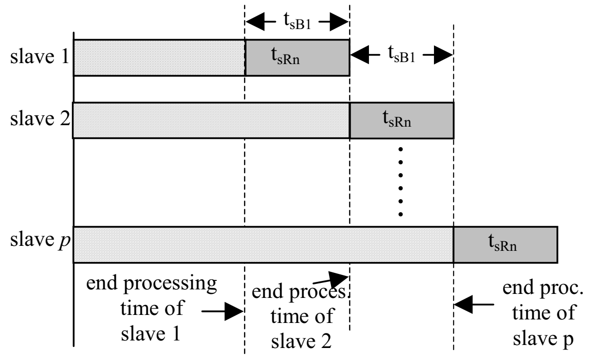
t_{sB1}
slave 1
t_{sRn}
t_{sB1}
slave 2
t_{sRn}
slave p
t_{sRn}
end processing time of slave 1
end proces. time of slave 2
end proc. time of slave p

Fig. 6. Processing/collection time scheme of the last result message.

The delay with which slave i finish its computation with regard to slave i-1 is the same than the delay with which slave i begins its computation with regard to slave i–1.

As the communication time of first data message is given by (3) or (4), if l_n is the number of result lines delivered by the processor i, in the case that the network fragments messages, the following inequality must be satisfied:

$$\left\lceil \frac{(b+a)\cdot n}{N_{bp}}\right\rceil\cdot t_s+(b+a)\cdot n\cdot t_w \ge \left\lceil \frac{l_n\cdot RB\cdot n}{N_{bp}}\right\rceil\cdot t_s + l_n\cdot RB\cdot n\cdot t_w \tag{11}$$

Otherwise, the inequality to be satisfied is:

$$t_s+(b+a)\cdot n\cdot t_w \ge t_s+l_n\cdot RB\cdot n\cdot t_w \tag{12}$$

The second condition we have taken into account for calculate the size of the messages of results is the following:

Condition 4: *Before the last message has been sent, the remainder of the results must be collected using the minimum number of messages, but taking into accounts that:*

4-1. *The processors must compute these results before sending them.*

4-2. *When the first processor is computing the results of block **i** all the processors must send, in a sequential way, the results of block **i-1**.*

Next figure gives a representation of previous condition.

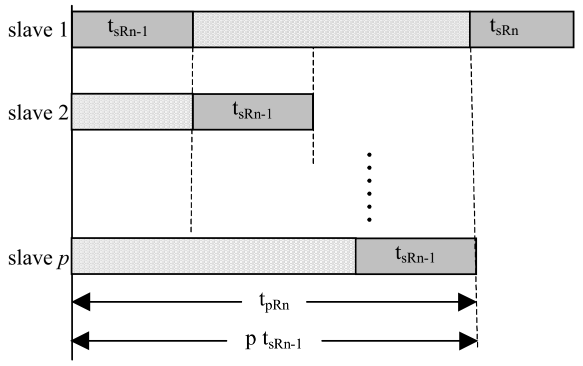

Fig. 7. Processing/collection time scheme of middle result messages.

As we have pointed out previously, the number of messages must be minimised because of the overhead of t_s. This implies that new data have to be received by all the slaves in a time equal or lower than the processing time a slave needs for processing data received in the previous message.

So, in the case that the network fragments the messages, the following inequality has to be satisfied:

$$l_i \cdot n \cdot t_c \geq p\left(\left\lceil {l_{i-1} \cdot RB \cdot n}/{N_{bp}} \right\rceil \cdot t_s + l_{i-1} \cdot RB \cdot n \cdot t_w\right) \quad (13)$$

Otherwise, the inequality to be satisfied is given by:

$$l_i \cdot n \cdot t_c \geq p\left(t_s + l_{i-1} \cdot RB \cdot n \cdot t_w\right) \quad (14)$$

VI. RESULTS

From the simulations we have carried out, some advantages have been deduced of optimising data distribution, minimising the time with which the slaves receive the first data and minimising also the time they wait for the last slave to send the data.

We have analysed eight different models, combining four basic models (Table I) with two number of slaves' processors p (4 or 16). Basic models are selected depending on the size of the image $m \times n$ (640×320 or 2000×4500) and the algorithm parameters a (2 or 8) and t_c (5 µs or 100 µs). For comparative purposes, we also show the time T_{seq} taken by a sequential processor to process each algorithm.

Tables II, III and IV show the experimental results obtained from the analysis of the models. We consider three different interconnection networks: Fast-Ethernet (Table II; t_s=700 µs; t_w=80 ns; N_{bp}=1500), Gigabit-Ethernet (Table III; t_s=150 µs; t_w=8 ns; N_{bp}=1500), and Myrinet (Tables IV and V; t_s=5.71 µs; t_w=2 ns).

Columns third and fourth of Tables II to V corresponds to:

- Theoretical number of message packets that can be collected per processor (N_{tpt}).
- Real number of message packets collected per processor (N_{tpr}).

Finally, the three last columns give the information that corresponds properly to the proposed algorithm and the improvements with regard to traditional systems:

- Time taken by the proposed parallel system to process the algorithm (T_{par}).
- Communication time improvement with regard to traditional systems ($T_{improv.}$).
- Percentage improvement of the proposed system over traditional systems (*Improv.*).

To evaluate the improvements provided by the proposed system, we have considered that the time used by the traditional system is, according to Fig. 8, the time first slave needs to receive all its data (t_{sD}). Then, while this slave processes its data the master continues sending data to the remainder slaves of the system.

Finally, first processor must transmit its results to the master and it waits till the other processors perform their transmissions. The time used by the system has been deduced from the time so obtained.

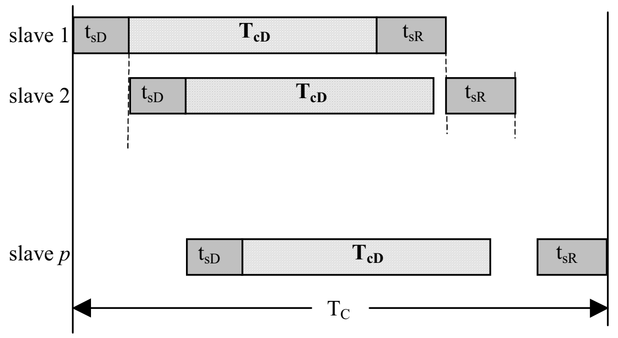

Fig. 8. Communication/processing time scheme on traditional system.

So, if the network fragments the messages, the time first slave needs to receive all its data is given by:

$$t_{sD} = \left\lceil {(n \cdot m)}/{(p \cdot N_{pb})} \right\rceil \cdot t_s + \left({(n \cdot m)}/{p}\right) \cdot t_w \quad (15)$$

Otherwise:

$$t_{sD} = t_s + \left({(n \cdot m)}/{p}\right) \cdot t_w \quad (16)$$

And the time taken by a slave for computing its data is given by:

$$T_{cD} = \left({(n \cdot m)}/{p}\right) \cdot t_c \quad (17)$$

If the network fragments the messages, the time taken by a slave to send a result message will be given by:

$$t_{sR} = \left\lceil {(n \cdot m \cdot RB)}/{(p \cdot N_{pb})} \right\rceil \cdot t_s + \left({(RB \cdot n \cdot m)}/{p}\right) \cdot t_w \quad (18)$$

Otherwise:

$$t_{sR} = t_s + \left({(RB \cdot n \cdot m)} \big/ {p} \right) \cdot t_w \qquad (19)$$

So, the total time we consider that a traditional system needs to process the algorithm is:

$$T_c = t_{sD} + T_{cD} + p \cdot t_{SR} \qquad (20)$$

And then, the improvement time will be as follows:

$$T_{improv.} = T_c - T_{par} \qquad (21)$$

The results we have presented correspond to the models that satisfy conditions 1-1 and 1-2. As it was expected, it has been verified that there are more considerable improvements for slow networks than for quick ones. It has also been verified that improvements are almost independent of the images' size.

We must emphasise that the improvement is more notable if the per-pixel computation time of the algorithm is small, what corresponds to those that must to be processed in real time.

We have also observed that it is very interesting to study the cases wherein the computation time is lower than transmission and results reception times.

Other important conclusion is that as great the number of slaves is, the great is the improvement.

Observing Table V we can deduce that for very quick networks, such as Myrinet, if the per pixel communication times are small the proposed system can be very advantageous.

VII. Conclusions

The system shows a very good behavior using slow interconnection networks and large number of processors. It can be seen also that for applications that must be executed in real-time, where they need a big amount of processors and have short per-pixel computing time, is when the system shows the best performance.

It is necessary to continue studying the cases with larger times of distribution than computation, and also the applications where the results are bigger than a byte per pixel.

References

[1] A. Downton and D. Crookes, "Parallel architectures for image processing", *Elect. & Comm. Eng. Jour.*, vol. 10, pp. 139-151, 1998.

[2] M. Hamdi and C. K. Lee, "Adaptive load-balancing of image processing applications on clusters of workstations", *IEEE trans. On Computer*, vol. 39, no. 10 pp.1477-1232, 1997.

[3] C. H. Cap and V. Strumpen, "Efficient parallel computing in distributed workstations environments", *Parallel Computing*, vol. 19, pp. 1221-1234, 1993.

[4] R. Steifert, "Gigabit Ethernet: technology and applications for high speed LANs", Addison-Wesley, pp. 141-280, 1998.

[5] Myrinet, http://www.myri.com/myrinet/overview/ , 2004.

[6] F. Bonomi and A. Kumar, "Adaptive optimal load-balancing in a nonhomogeneous multi server system with a central job scheduler", *Parallel Computing*, vol. 19, pp. 1221-1234, 1993.

[7] S. Miguel and Y. Robert, "Adaptive load-balancing for image processing algorithms", *Parallel Computation Proc. 1st international*, pp. 438-451, 1991.

[8] Message Passing Interface Forum, "MPI-2: A Message Passing Interface Standard", *High Performance Computing Applications*, 12(1-2):1-299, 1998.

[9] RDMA Consortium, http://www.rdmaconsortium.org/ , 2004

[10] F.J. Seinstra and D. Koelma, "Incorporating Memory Layout in the Modeling of Message Passing Programs", *Proc. Of 10th EUROMICRO-PDP'02*, Canary Islands, Spain, pp. 293-300, September 2002.

TABLE I
Simulation Models

	Image		Algorithm					
Model	*m*	*n*	*a*	*tc*	*Tseq*	*tc'*	*Tseq'*
1	2000	4500	2	5 μs	45 s	*100 ns*	*900 ms*
2	640	320	2	5 μs	1.024 s	*100 ns*	*20.5 ms*
3	2000	4500	8	100 μs	900 s	*500 ns*	*4500 ms*
4	640	320	8	100 μs	20.480 s	*500 ns*	*102.4 ms*

TABLE II
Results Obtained for Fast-Ethernet

Model	*p*	*Ntpt*	*Ntpr*	*Tpar* (ms)	*Timprov* (ms)	*Improv*
1	16	122				
1	4	3503	3006	11289	6110.64	54.13 %
2	16	2				
2	4	79	74	262.21	136.77	52.16 %
3	16	4893	774	56604	4873.26	8.61 %
3	4	79842	3024	225089	6061.44	2.69 %
4	16	111	20	1306.1	98.42	7.54 %
4	4	1816	72	5126.5	136.46	2.66 %

TABLE III
Results Obtained for Gigabit-Ethernet

Model	*p*	*Ntpt*	*Ntpr*	*Tpar* (ms)	*Timprov* (ms)	*Improv*
1	16	1111	756	2843.6	1001.65	35.22 %
1	4	18510	3006	11256	1207.22	10.73 %
2	16	25	21	69.046	19.65	28.45 %
2	4	421	72	256.63	27.04	10.54 %
3	16	23377	774	56320	962.77	1.71 %
3	4	374760	3024	225017	1197.50	0.53 %
4	16	531	20	1285.2	19.52	1.52 %
4	4	23377	774	56320	962.77	1.71 %

TABLE IV
Results Obtained for Myrinet with *tc*

Model	*p*	*Ntpt*	*Ntpr*	*Tpar* (ms)	*Timprov* (ms)	*Improv*
1	16	30390	4	2813	19.22	0.68 %
1	4	490980	4	11250	22.53	0.2 %
2	16	691	4	64.122	0.53	0.83 %
2	4	11172	4	256.03	0.54	0.21 %
3	16	615302	4	56251	19.22	0.03 %
3	4	9849562	4	225000	22.53	0.01 %
4	16	224132	4	5120	0.54	0.01 %
4	4	14001	4	1280.2	0.53	0.04 %

TABLE V
Results Obtained for Myrinet with *tc'*

Model	*p*	*Ntpt*	*Ntpr*	*Tpar* (ms)	*Timprov* (ms)	*Improv*
1	16	221	9	56.92	19.22	33.77 %
1	4	8274	8	225.13	22.53	10.01 %
2	16	5	7			
2	4	188	7	5.15	0.54	10.5 %
3	16	2684	5	282.64	19.22	6.80 %
3	4	476792	5	1125.3	22.53	2.00 %
4	16	1084	5	25.65	0.54	2.11 %
4	4	61	5	6.59	0.53	8.07 %