Parallel image quantization using LAN of workstations

Stéphane Ubéda, Xavier Vigouroux

Parallel image quantization using LAN of workstations

Stéphane Ubéda
Xavier Vigouroux

July 10, 1995

Technical Report N° 95-01

Parallel image quantization using LAN of workstations

Stéphane Ubéda
Xavier Vigouroux

July 10, 1995

Abstract

This report presents the work done to parallelize a quantization algorithm of color images on a LAN of workstations. The quantization consists in reducing the number of colors in order to reduce the length of an image. This problem belongs to the NP-complete problems (comparable to the construction of an optimal decision tree). The communications between the different parts of the system have been managed by PVM.

One important point is the use of the distributed storage that represents a set of workstations. This distribution allows the complete parallelization of the Read/Write operations. Furthermore an interface has been realized to make this parallel algorithm available to any kind of people.

Keywords: Image processing, quantization, parallel processing

Résumé

Ce rapport présente le travail fait pour paralléliser l'algorithme de quantification d'images couleur sur réseau local de stations de travail. La quantification consiste à réduire le nombre de couleurs afin de réduire la taille de l'image (en nombre d'octets). Ce problème appartient à la classe des problèmes NP-complet (comparable à la construction d'un arbre de décision optimal). Les communications entre les différents éléments constitutifs du système ont été gérées par PVM.

Un point important est l'utilisation des unités de stockage distribuées sur chacune des stations de travail. La distribution permet la complète parallélisation des phases d'Entrée/Sortie. De plus, une interface a été réalisée pour permettre à tout type de personnes d'utiliser cet algorithme parallèle.

Mots-clés: traitement d'images, quantification, traitement parallèle

1 Introduction

The aim of color quantization is to reduce the number of colors in an image according to a specific criterion. An usual criterion is the conservation of the aspect of the remaining picture according to visual human perception. Effectively, after such a preprocessing, the different algorithms applied to the image are all the more efficient the some noises are reduced. A lot of studies have been done to succeed in the research of a set of colors that is smaller than the original set and such that the new image is as "close" as possible to the original one.

Obviously, this is a problem of minimization of an energy function and thus the resolution of this problem is NP-complete. The classical method to solve such problems is to use an heuristic that leads to a nearly optimal solution. We present here one of these heuristics which is sequential and then transcript it to make it parallel.

First, let us explain the sequential formalism and the algorithm.

1.1 Formalism

This formalism (and the sequential algorithm) is extracted from [WPW90]. Its advantage is to be simple, and nevertheless powerful. Let $\Omega = \{(r,g,b) | 0 \leq r,g,b \leq 255\}$ be the *RGB color space*. We choose this color space among the different ones (see [Poy94] for a list, or [GW93, Page 227]) for the sake of simplicity. Furthermore, the information is distributed among each axis without predilection; the algorithms using this space are therefore much more simple. Let $(x,y) \in I \times I$ be the coordinates of the pixels in the original image. I is an integer set. A digital image is defined as a mapping which assigns a color to each pixel:

$$h : I \times I \rightarrow C \subset \Omega \tag{1}$$

where $C = \{\vec{c_1}, \vec{c_2}, \ldots, \vec{c_N}\}$ is the palette and N is the total number of colors in the original image. A quantized image is defined as a mapping which assigns a new color to each pixel:

$$f : I \times I \rightarrow R \subset \Omega \tag{2}$$

You can notice that R is not necessarily a subset of C: $R = \{\vec{r_1}, \vec{r_2}, \ldots, \vec{r_K}\}$ with $K \ll N$.

The replacement of a color $\vec{c_i}$ by a representative color $\vec{r}(\vec{c_i}) = (r', g', b') \in R$ will generate an error defined as:

$$||\vec{c_i} - \vec{r}(\vec{c_i})||^2 = (r - r')^2 + (g - g')^2 + (b - b')^2 \tag{3}$$

Now, we can compute the total error on the quantization of an image of size $||I|| \times ||I|| = M^2$ with

$$D(h,f) = \frac{1}{M^2} \sum_{(x,y) \in I \times I} ||h(x,y) - f(x,y)||^2 \tag{4}$$

If $\forall i \in \{1, \ldots, N\}, p(\vec{c_i})$ denotes the relative occurrence frequencies of the colors in the original image. The last equation may be rewritten as

$$D(h,f) = \sum_{i=1}^{N} p(\vec{c_i}) ||\vec{c_i} - \vec{r}(\vec{c_i})||^2 \tag{5}$$

Obviously, in the set of colors R, we choose the best color $\vec{r_j}$ to substitute an original color $\vec{c_i} \in C$. The best color means the color that minimizes the equation 3. with these definitions and notations, the problem of quantization may be expressed as:

> *the objective of quantization is to find a set of K colors to substitute with the best accuracy the N colors ($N \gg K$) of an original image. Still according to [WPW90], this problem is at least as difficult as the construction of an optimal binary decision tree. This latter is known as to be NP-Complete, thus the former problem too.*

Now, let us see the sequential algorithm that leads to a parallel algorithm.

1.2 Sequential algorithm [WPW90]

The idea is to construct, for the original image, the 3D histogram of the occurrence frequency of each color. In other terms, we construct a RGB-cube that has in the box (r_0, g_0, b_0) the number of pixels with this color. At once this cube is constructed, the algorithm consists in creating a set of boxes that will be associated with one and only one color: **the representative color**. Different solutions exist to get a color from a box $(r_1, g_1, b_1) - (r_2, g_2, b_2)$:

- first, we have to choose the color that will represent the box. We can take
  - the mean of the colors $(\frac{r2-r1}{2}, \frac{g2-g1}{2}, \frac{b2-b1}{2})$,
  - the median,
  - . . . ;
- then we have to choose the part of the RGB-space that will be reduced to this color
  - the box itself,
  - the element of the 3D Voronoi partition of the space,
  - . . . .

The solutions are numerous. We test different solutions and the result is always (perceptibly) similar. After having seen the way to make correspond an original color to a representative color, the table 1 shows how to construct these boxes.

This algorithm is very easy to implement on a sequential computer. But, on a parallel computer, the amount of data to exchange is prohibitive. Thus, the parallel version is slightly different than the sequential one; generally, it has been simplified.

2 Distributed Algorithm

2.1 Model

A Local Area Network of workstations is not at all a parallel machine. This is a set of processors, but the communication network is a bus. First, it implies that there is only one communication at a time; second, there is no topology. The model is thus clear, the global communication time is the sum of all the communication times.

Usually, we consider the cost (in time units) of communications as a $\beta + l.\tau$ function. l is the length of the message (in information units), τ is the "bandwidth" (ie. the number of information

```
1. For the number of wanted Colors do
   1.1 Select the sub-box with the largest variance (the one which division will separate the larger
       number of points in the RGB-Space),
   1.2 For Each Dimension do
       1.2.1 Project the distribution according this dimension,
       1.2.2 Select the optimal threshold that will create the most different two sub-boxes (with
             the most different representative color),
       1.2.3 Compute the weighted sum of the variance of the two intervals
   1.3 The division plane is the one that is perpendicular to the axis with the minimum weighted
       sum of projected variances and passes through the optimal threshold.
   1.4 Divide the box into two sub-boxes.
```

Table 1: Sequential algorithm

units per time units) and β is the startup time due to the buffer allocation. Here, this model is enhanced by another term in the sum because the startup time is not a constant but also depends on the length of the message. Indeed, the buffer allocations seems to be implemented as a linked list of small buffers, thus the buffer construction is built up from several parts. This implies that the communication costs is now $\beta_1 + \lfloor l/size(small\ buffer) \rfloor \beta_2 + L\tau$

2.2 PVM

We decide to use PVM with Sun WorkStations. PVM (stands for "Parallel Virtual Machine") [BDG+91, BDG+92, GS91] is a widely used package with two main properties: portability and heterogeneity. It can be installed and used by anybody without any privileges. It can be used on one workstation as well as on a set of parallel machines. All these points make PVM as attractible for a newcomer in parallel processing, as for an expert: they can use their usual programming environment to test their program before execute them on one of the architectures they are used to.

2.2.1 First tries to measure communication times

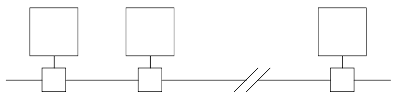

Figure 1: Communication layout with PVM

As we have already written, the problem of PVM on a LAN (Local Area Network) of workstations[1] resides in the network itself. First, the network is a shared bus, thus only one communication takes place at a time on the entire network. The second problem is that there is no topology. Thus

[1]and even more for a WAN (Wide Area Network)

no optimization may be done to make the communications faster. These remarks lead us not to optimize the communication phase, because it would be a lack of time. The communication layout is therefore very simple (see figure 1). This point is very important for the design of the algorithm, because we must minimize the communications in the system not to make them sequential.

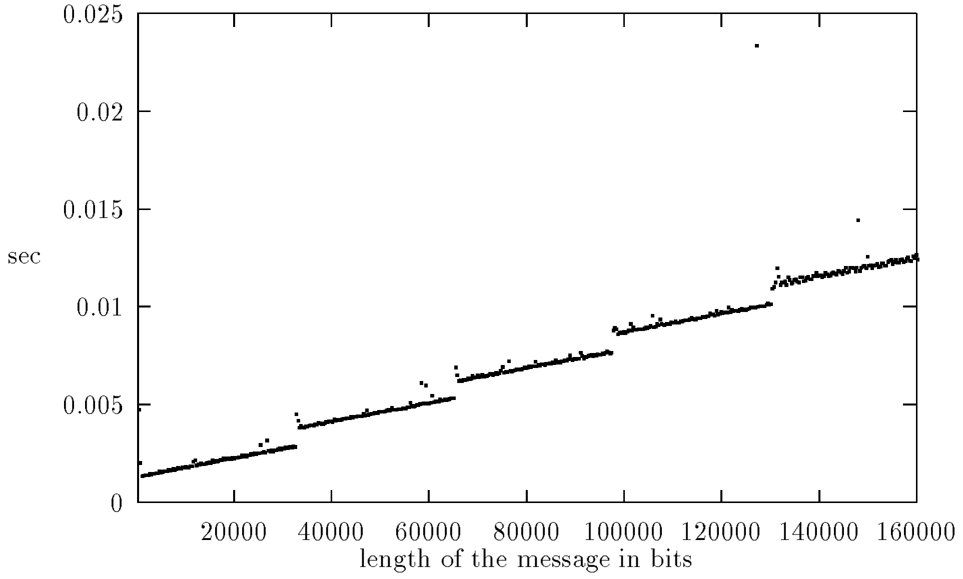

Figure 2: Communication time with PVM 3.3.7 and Solaris 2.4 (option DataDefault)

The figure 2 shows the internal behavior of PVM by displaying evidently the buffer allocation. Each buffer size is 32768 bits (ie. 4096 bytes).

$$time \;=\; \beta_1 + \lfloor l/32768 \rfloor \beta_2 + L\tau \tag{6}$$

where

$$\begin{aligned} \beta_1 &\approx 1200\,\mu sec \\ \beta_2 &\approx 1000\,\mu sec \\ \tau &\approx 0.04882\,\mu sec.b^{-1} \Rightarrow 20\ Mbit.s^{-1} \end{aligned}$$

Be careful that τ, is not the band width because the time to send 20 $Mbit$ according to equation 6 is not 1 second but a lot more: 1.6 second.

If we want to use a simple evaluation of the communication time, we can try to find the line crossing the middle of each segment. The equation of such a line is $y = 0.074x + 812$ where x unit is bits and y μsec. The band width is then more realistic with 13.5 $Mbits.s^{-1}$. This value is nevertheless too large.

The figure 3 is the same test with the same conditions but at a different moments (few seconds after the figure 2). It shows how the behavior of PVM may change with the network occupation. The band width is 6.5 $Mbits.s^{-1}$ and the startup 1500 μsec.

The communication time of the figure 4 shows the behavior of PVM when the programmer use the `DataInPlace` strategy. First, this strategy avoids to copy the buffer to send; and second, it

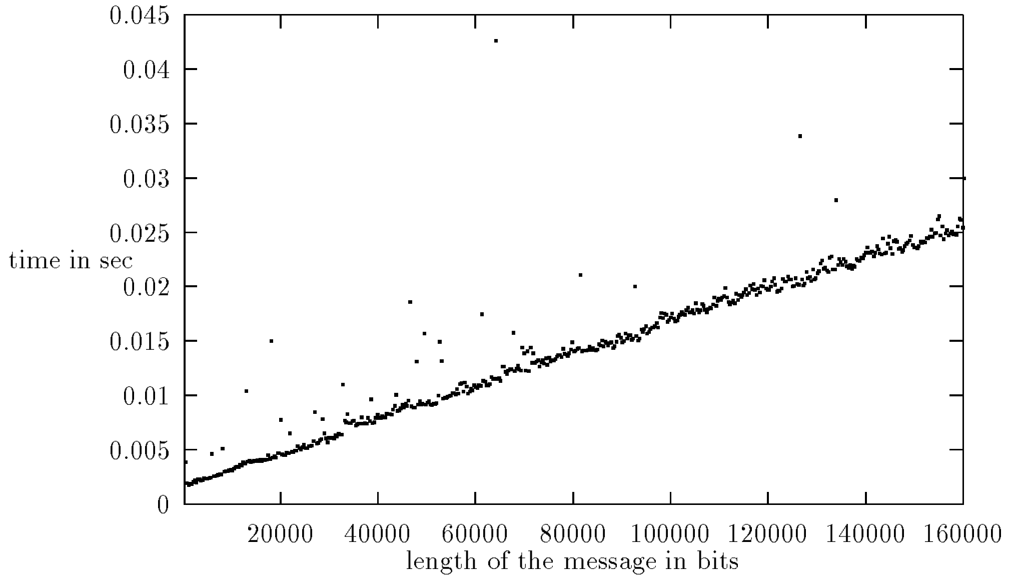

Figure 3: Communication time with PVM 3.3.7 and Solaris 2.4 (option DataDefault) with network load

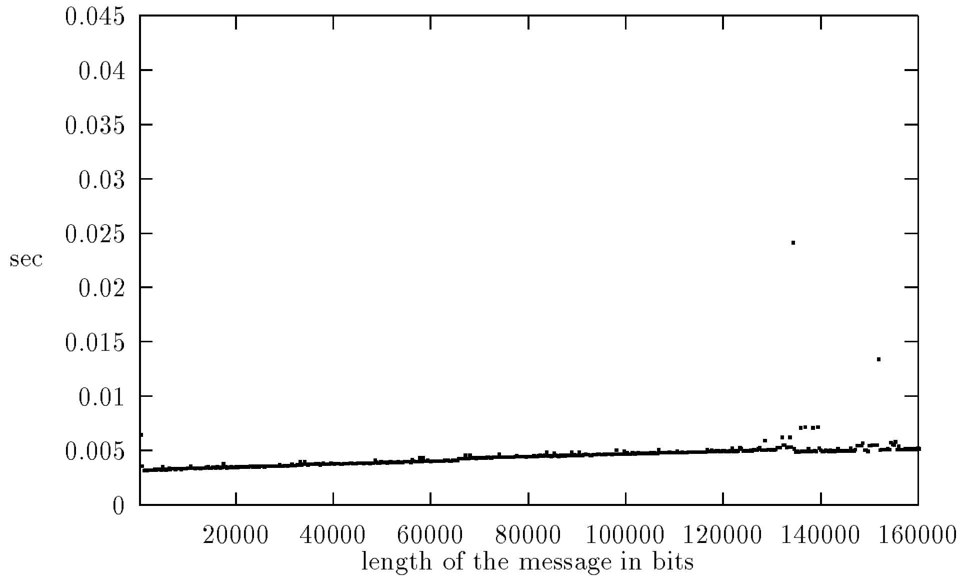

Figure 4: Communication time with PVM 3.3.7 and Solaris 2.4 (option DataInPlace)

does not use the XDR library to encode the data. This strategy is worse than the first one as long as the size of the message is greater than 1024 integers (ie. 4096 bytes). The formula expressing the communication time is then

$$time \;=\; \beta_1 + \lfloor l/65536 \rfloor \beta_2 + L\tau \tag{7}$$

where

$$\begin{aligned} \beta_1 &\approx 3000\,\mu sec \\ \beta_2 &\approx 100\,\mu sec \\ \tau &\approx 14\,\mu sec.bit^{-1} \Rightarrow 71\ Mbit.s^{-1} \end{aligned}$$

this band width is larger than the theoretical maximum Internet band width.

2.2.2 Real Communication times

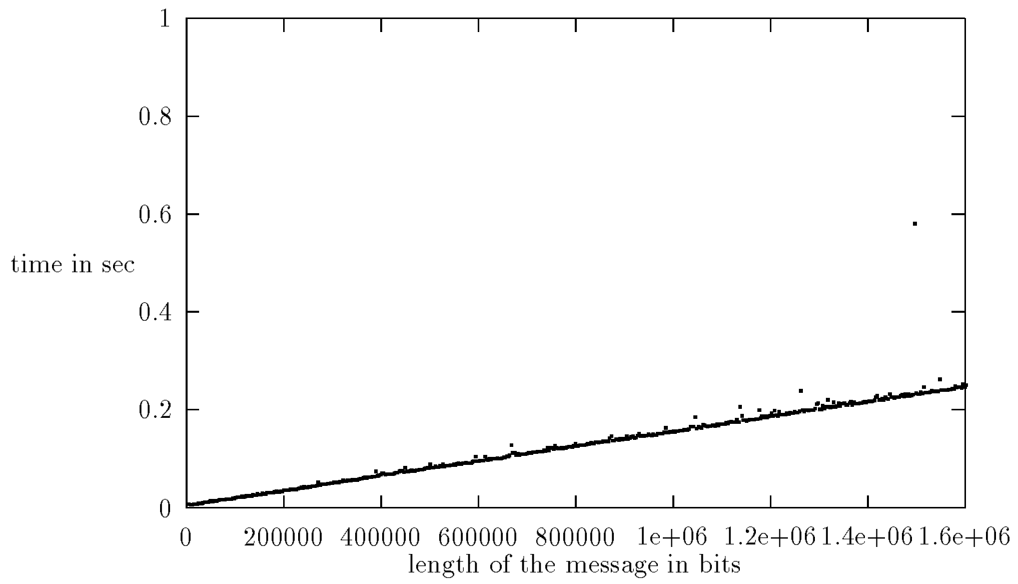

Figure 5: Communication time with PVM 3.3.7 and Solaris 2.4 on very large messages

These values are greater than the Internet theoretical band width: $10 Mbits.s^{-1}$. In fact, the size of the messages makes this result not significant (they are too short). When we choose a larger scale, we can see that the band width is not larger than the theoretical but rather $6.5 Mbit.s^{-1}$ (see figure 5) in the standard case and $8 Mbit.s^{-1}$ (see figure 6) with the option `DataInPlace`..

2.2.3 Resulting choices and algorithm

The central idea of the distributed algorithm is the same as the sequential one, but two things must be distributed: the data and the computation.

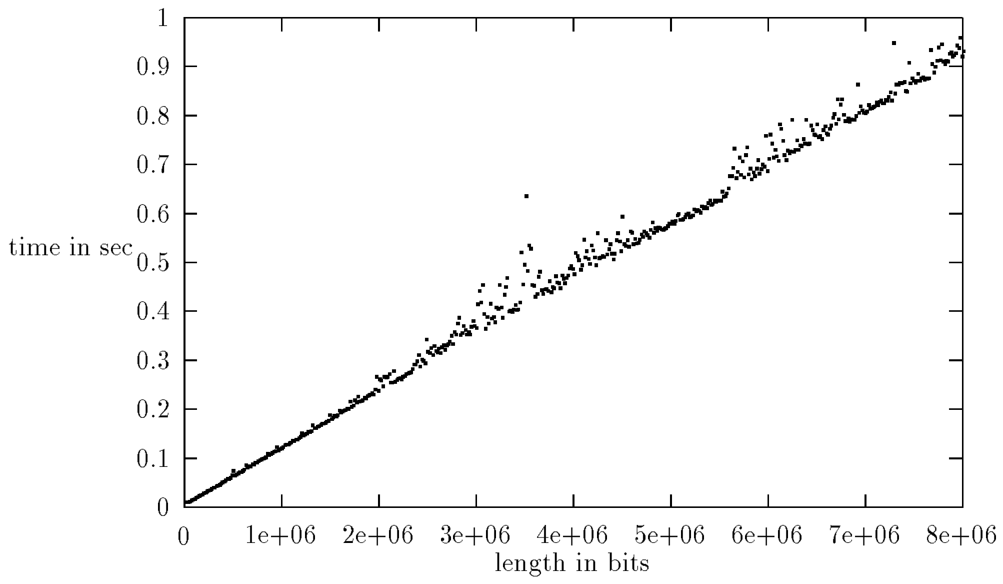

Figure 6: Communication time with PVM 3.3.7 and Solaris 2.4 (option DataInPlace) on very large messages

- concerning the data, the image is distributed on the different processors, and even on different hard disks. Effectively, using PVM on workstations makes it possible to use their disks. This way, the load phase, which is very costly for image processing problems, becomes lighter. The distribution is made along horizontal blocks (see figure 7).
- the computation is also distributed: each processor computes one part of the box division.

2.3 Parallel I/O

As we have already said in the previous section, the workstations offer different sites where to put data. Thanks to this, we distribute the image on the different hard disks. We choose to divide the image horizontally because of the fact that a scene is usually divided this way. For instance, the sky is at the top, the ground at the bottom, ... Then, the number of different colors on each machine will be reduced.

At the end of the treatment, the image slice is also stored on the local disk. A merging phase is then added to the execution to create the global quantized image. We test the speed of the Read/Write operations to see what was the behavior of the remote disk operation. Figure 8 shows the result of this test.

2.4 Data structure

Three structures are very important in this algorithm: the RGB-Cube, the box and, obviously, the image.

- **The cube** is a 3D array of integers (see figure 9). Each cell contains the number of pixels with this color. The problem of this cube is that it may be very spare.

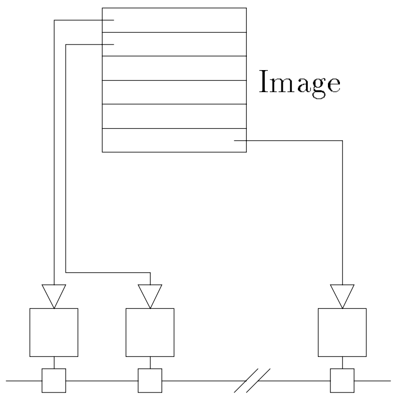

Figure 7: Image division

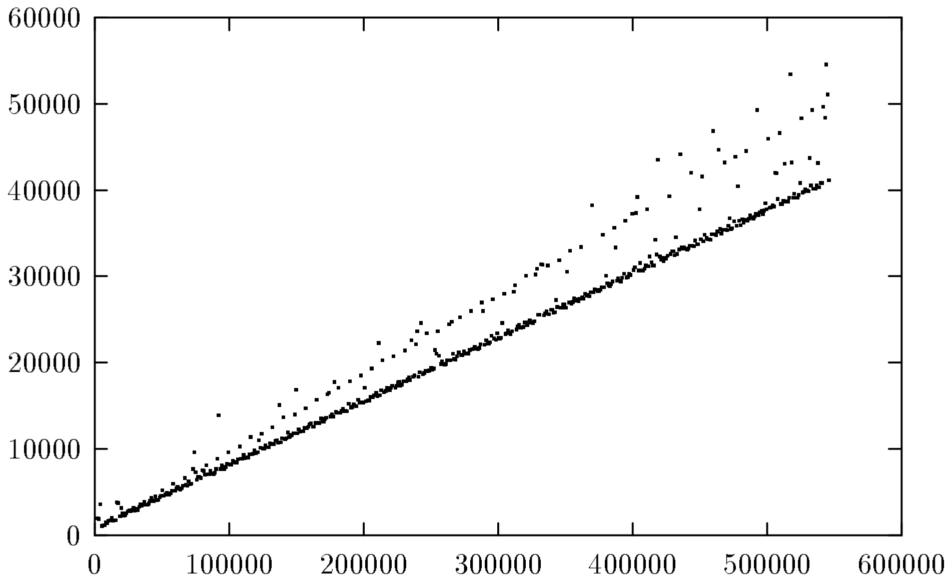

Figure 8: Read operation on remote host with PVM 3.3.3 and SunOs 4 (μsec as a function of file length)

The first optimization consists in reducing the number of bits assigned to each colors: for instance, if each color is coded on 8 bits in the original image, we can choose a cube with only 5 bits. Then each cell is no more 1 color but $(2^{8-5})^3 = 512$.

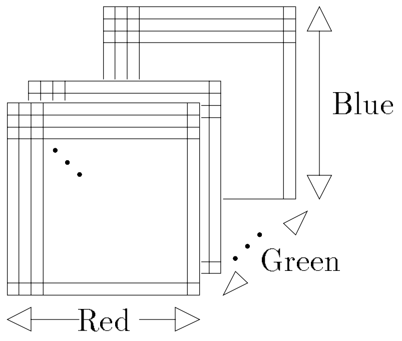

Figure 9: Cube structure

The second optimization we choose to reduce the amount of empty cells, is to make local bounds (see figure 10):

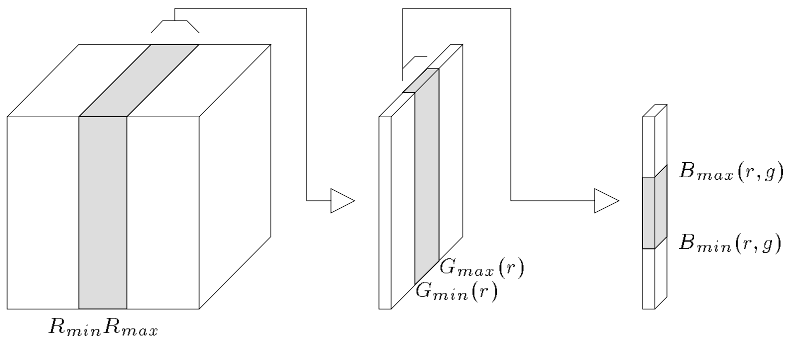

Figure 10: Example of cube compression

– the red dimension is limited by a R_{min} and R_{max},

– each plan $r \in [R_{min}, R_{max}]$ as green local bounds $G_{min}(r)$ and $G_{max(r)}$,

– Finally each line $r \in [R_{min}, R_{max}]$ and $g \in [G_{min}(r), G_{max}(r)]$ has blue local bounds $B_{min}(r,g)$ and $B_{max}(r,g)$.

The domain of computation is then greatly reduced. A test with a 5 bits RGB-cube has shown a gain of 85% !!! The order in which we consider the color must not have any consequence in the compression rate because the information is equally distributed amongst the three axes. This way, in mean, the results must be the same with the six different axes enumeration: RGB, RBG, GBR, GRB, BRG, BGR. beside this remark, the compression rate is not the best we should obtain because of this RGB space; see section 4.2.2 for further information.

Lastly, the third optimization, is to consider a cell with the value 1 as a void cell. This optimization increase the performance of the second one. But, this optimization should reduce the quality of the produced image ; in fact, the result is not (perceptibly) different.

- **the Box**: once the cube of frequencies is constructed, a box only consists of three intervals: one for red, one for green and one for blue. The frequencies are shared with the cube; thus, the information is not repeated. Other fields are added to make it possible to compute the necessary information for the segmentation algorithm. Typically, we have the projection along each direction (sum, weight, array of projected values) and the mean.
- **the Images**: the source image is a raster file with 8 bits per color, the resulting image is also written in such a format.

2.5 Communication with the RGB-cube

As we have seen in the previous section, the cube is optimized not to send large message and thus to reduce the communication cost. In practice, we exchanged only the interesting data (ie. a subset of each color interval). This optimization has shown its efficiency by greatly reducing the communication cost. As we have to merge the cube, we could think that a merging binary tree would have been the best solution (see figure 11). With PVM on a set of workstations, this is a worse solution than merging everything on a single processor.

First, the number of communications is larger in the tree. We have seen the communications are sequential on a set of workstations, thus the number of communication corresponds to the time spend by the global system. Let us count the number of communications int the both cases.

- the case of the tree: if p represents the number of processors, then, whatever the shape of the merging tree, the number of internal nodes is equal to $p-1$. As each internal node needs two communications (for the both entries), the number of communications is equal to $2(p-1)$,
- the case of the local merging: the number of communication is obviously $p-1$.

Furthermore, the messages length in the first case is larger than in the second, because, as we merge the data, the cube becomes larger and larger. Intuitively, the better is the linear merge.

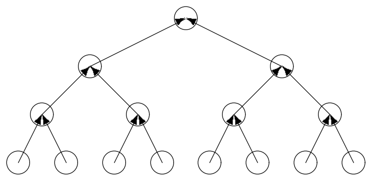

Figure 11: Example of merging binary tree

The algorithm is now clear (see table 2).

2.6 Who computes What

For the sake of communication reduction, we do not want to have more exchanges than two broadcasts (one for image, one for cube). This implies during the color computation that no processor

```
1. distribute the source image if necessary
2. distribute the parameters (size of the cube, number of wanted colors)
3. For each Processor Do
    3.1 compute the RGB-cube
    3.2 send it to the master who merges it
    3.3 receive the complete one
    3.4 compute a part of the colors (ie. some boxes)
    3.5 send them to the master who merges them
    3.6 receive the complete set of colors
    3.7 compute the new palette
    3.8 apply it to the part of the image
4. merge the global quantized image
```

Table 2: Parallel Algorithm

exchanges data. Thus, every one must have a coordinated behavior. This is done by numbering each processor. Each processor only knows how many processors are in the machine and the number of wanted colors.

Let us choose one processor among the $p = 2^n$. Its number i written in binary is $i_0 i_1 \dots i_{n-1}$. This number is the code of the tree branch to compute: each box is divided into two sub-boxes. The lowest is the number 0 and the highest the number 1. Obviously, several processors compute the same part, but at the end, they conclude by a unique calculus (fig 12)

The problem of this method comes from the fact that some boxes may become void. Then, it is impossible to divide them. Thus, the number of colors a processor obtains is less than the expected number. Furthermore, it is very difficult to avoid this problem without exchanging a lot of data (all to all) In fact, this problem is not very important because if a box can not be divided, it implies that no division is really needed, so such a division would have been with no effect on the resulting image.

3 Performances

First, the different optimizations on the structures lead to very good performances (even in sequential). For instance, on a single processor, the time to execute the program on a 600×303 image with a 6 bits cube and a 64 colors palette is 5 sec on a Sparc5!!!

But, as the computation is very sharp, the communication cost is very important in the total time, thus, with 2 hosts the total time is 4.5 sec.

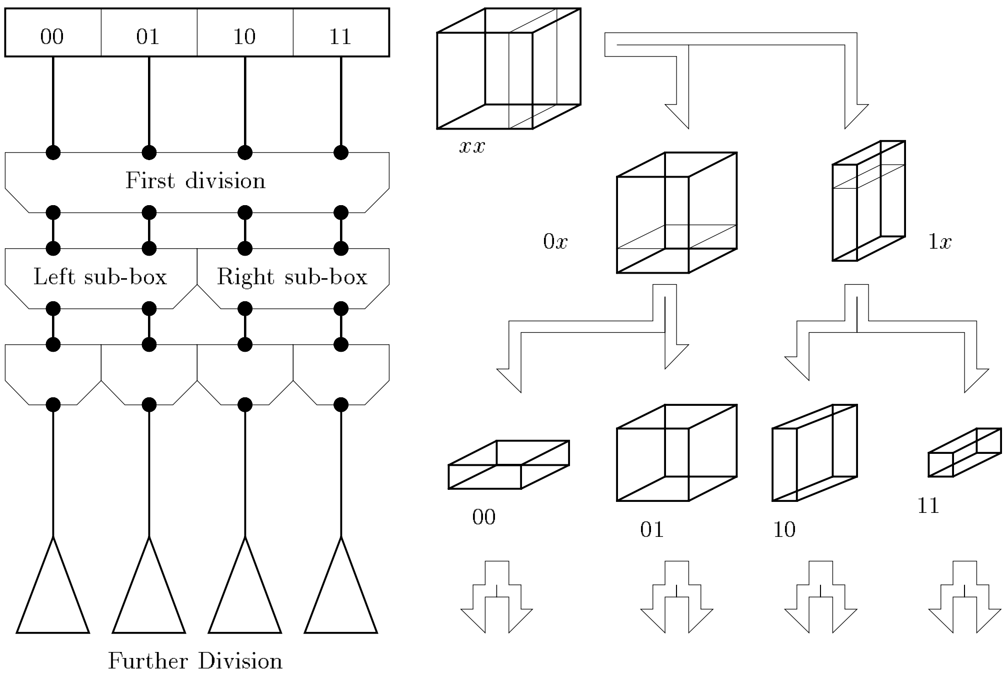

Figure 12: Example of Work repartition with 4 processors

Number of hosts	time (sec)			
	5 bits		6 bits	
	64 cols	128 cols		64 cols
1	1.6	3.6	5.5	5
2	1.4	2.3	4.9	4.5
4	1.6	2.7	7.8	4.7

Table 3: different execution times

4 Conclusion and Future work

4.1 Interface

This algorithm has been implemented to make it easy to use. We designed a window to manage easily the distributed program: image split and merge, processes spawn, ...

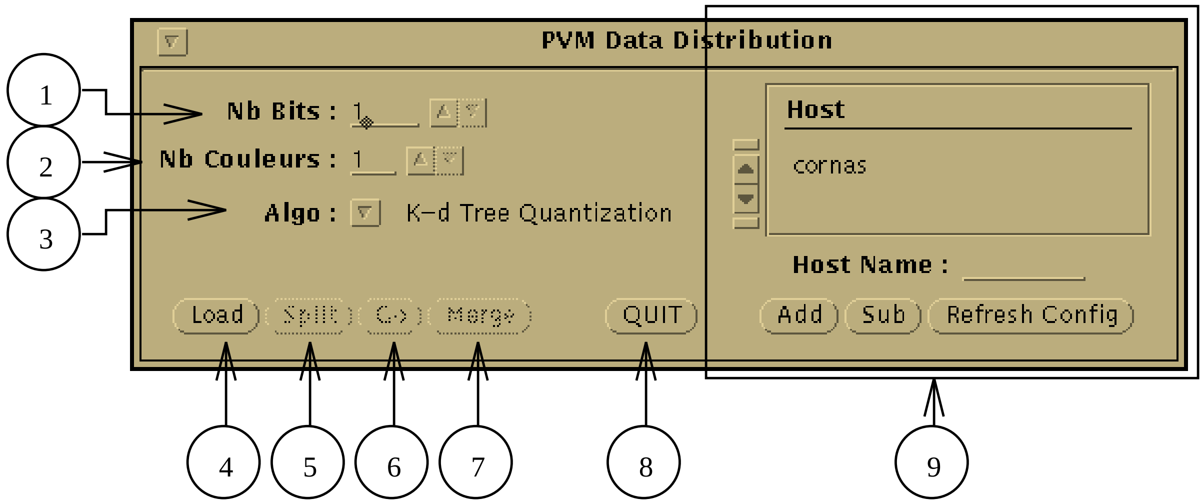

Figure 13: View of the Software Interface

In fig 13 :

1. the number of bits assigned to each dimension in the RGB-cube,
2. the number of expected colors,
3. the algorithm the user wants to use (at this time, there is only one, but others will be added),
4. $\boxed{\text{load}}$ is used to load an image,
5. $\boxed{\text{Split}}$: When the image is loaded you can split it among the different hosts,
6. $\boxed{\text{Go}}$ must be pushed to execute the algorithm,
7. $\boxed{\text{merge}}$ allows to merge the resulting slices of image,
8. $\boxed{\text{quit}}$ behavior is obvious,
9. This square is used to make the link with pvm. It allows to add a new host, or to suppress it, or to refresh the configuration list.

4.2 Future Work

4.2.1 The error criterion

The error we have chosen (see Sec. 1.1), in the RGB space, has not a perceptive reality but this is the most intuitive. Nevertheless [TCL94] explains that this error is insufficient because it does not take into accounts the spatial correlation between two adjacent pixels. We could define an error which is not a simple pixel to pixel comparison. For instance, if we take into account the difference between two adjacent pixels color we would be closer to the perception. Such an error may be defined by applying a 3×3 convolution mask on each pixel in both images and then making the difference between these two results. The mask may be as for the edge detection:

$$u_0 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad u_1 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad u_2 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

As the sum of the weights of these filters is 0, these filters make each pixel totally relative to its environment. By this way, one image a little bit more red than another will be very close to this last one.

4.2.2 The color space

The second optimization consisting of considering only the interval of value with non empty cells is inherently not the best one in the RGB space because of the cube itself. Effectively, a scene is composed of different objects. Each object will have its own color with different brightnesses due to the light reflection angles. Thus, in the RGB space, each object will create a segment of colors representing the color of the object with the different brightness. This segment, as we can see in figure 14 is not following one dimension but is rather parallel to the line $r = g = b$: $r = g+\alpha_g = b+\alpha_b$ (this is a simplistic formalization). And such a segment makes the optimization less efficient. If we had chosen another space like Hue-Saturation-Brightness or Hue-Lightness-Saturation the result would have been much better.

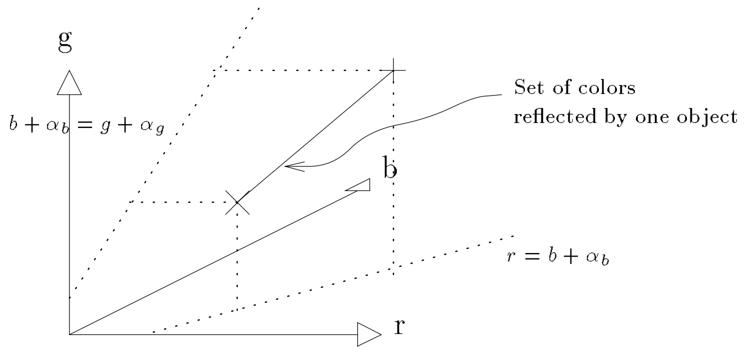

Figure 14: set of colors reflected by one object

4.3 Conclusion

The implementation of this algorithm has shown how the communications are costly with PVM on a LAN of workstations. However, the segmentation algorithm described in this paper must have a very good behavior on a multi-processor machine. But, we have not yet tried to translate it. Another future work concerns the implementation of different image processing algorithms with the same distribution methods (for instance, a Filter-Expectation-Minimization algorithm to segment a image).

References

[BDG+91] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, and Vaidy Sunderam. A users' guide to PVM (parallel virtual machine). Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, University of Tennessee, July 1991.

[BDG+92] Adam Beguelin, Jack Dongarra, Al Geist, Robert Manchek, Keith Moore, and Vaidy Sunderman. PVM and HeNCE : Tools for heterogeneous network computing. In Jack Dongarra and Bernard Tourancheau, editors, *Environments and Tools for Parallel Scientific Computing*, volume 6 of *Advances In Parallel Computing*, pages 139–154, Saint Hilaire du Touvet, France, September 1992. CNRS-NSF, Elsevier Science Publishers - North Holland.

[GS91] G. Geist and V. Sunderman. Experiences with network based concurrent computing on the pvm system. Technical Report ORNL/TM-11760, Oak Ridge Notional Laboratory, January 1991.

[GW93] Rafael Gonzalez and Richard Woods. *Digital Image Processing*. Addison-Wesley Publishing Company, 1993.

[Poy94] Charles Poyton. Frequently asked questions about colour. fetched at `Charles@mail.north.net`, 1994. Toronto, CAN.

[TCL94] A. Tremeau, M. Calonnier, and B. Laget. Color quantization error in terms of perceived image quality. In *IEEE Proc on Acoustics, Speech and Signal Processing, Adelaide, Australia*, 1994.

[WPW90] S.J. Wan, P. Prusinkiewicz, and S.K.M. Wong. Variance-based color image quantization for frame buffer display. *COLOR research and application*, 15(1), 1990.