**MVA '92** IAPR Workshop on Machine Vision Applications Dec. 7-9,1992, Tokyo

# AN EFFICIENT PARALLEL IMPLEMENTATION OF THE LAPLACIAN PYRAMID ALGORITHM

Min Xue, Abdelhamid Hachicha, and Alain Mérigot

ESIGETEL,

1, rue du port de Valvins, 77215 Avon Cedex, France

IEF, Université Paris-Sud,

Bât. 220, 91405 Orsay Cedex, France

## ABSTRACT

*The Laplacian pyramid coding is a useful image processing tool. Its running time with a sequential machine is $O(n^2)$ where n*n is the size of the original image. This paper describes a parallel implementation of this algorithm on SIMD machines like pyramids and meshes. In these machines, the concerned data and associated operations are allocated to elementary processors, and a well-organized scheduling is used to allow an evenly distributed computation executed in parallel. With a properly sized pyramid machine the running time is reduced to O(H) where H is the pyramid's height. When the machine is smaller than the Laplacian pyramid, the image is divided into segments and the computation is carried out segment by segment. However, a problem on the periphery of each segment is encountered. To remedy this problem, two methods are proposed. The best one allows a running time of $O(n^2/m^2 + \log m)$ where m*m is the size of the pyramid machine's base.*

## 1. INTRODUCTION

Our current project, carried out jointly by ESIGETEL and the Université Paris-Sud, concerns the implementation of image coding algorithms[6] with SPHINX[5], a multi-SIMD bin-pyramid machine designed by the Université Paris-Sud in France. This paper describes an efficient parallel implementation of the Laplacian pyramid algorithm[2]. This algorithm, presented by P. J. Burt and E. H. Adelson several years ago, is used for image compression, progressive image transmission, and other applications in image processing domain.

The reason of using a massively parallel machine is to reduce the running time of computation. In this paper, we are particularly interested in using a pyramid machine such as SPHINX and PAPIA[3], and in comparing them with meshes like MPP[1].

In the following sections, after a brief presentation of the Laplacian pyramid coding, we present in detail its parallel implementation followed by a performance evaluation.

## 2. LAPLACIAN PYRAMID CODING

The Laplacian pyramid coding consists of two main steps: the generation of a Gaussian pyramid and that of a Laplacian pyramid. Each layer of the Gaussian pyramid is a low-pass filtered image of its lower layer, except for the base which is the original image of size n*n. The difference between one layer of the Gaussian pyramid and the expanded image obtained from the layer immediately above gives the corresponding layer of the Laplacian pyramid. The expanded images form a pyramid called expanded Gaussian pyramid.

### Gaussian Pyramid

The Gaussian pyramid is the sequence of the reduced low-pass filtered images $\{g_k\}=\{g_0, g_1, ..., g_{H-1}\}$ generated from $g_0$ (the original image) by a REDUCE function:

$$g_{k+1}= \text{REDUCE}(g_k) \tag{2.1}$$

$$g_{k+1}(i,j) = \sum_{p=-2}^{2} \sum_{q=-2}^{2} w(p,q) g_k(2i+p, 2j+q) \tag{2.2}$$

where H is the height of the pyramid and w() is the weighting function.

### Expanded Gaussian Pyramid

The expanded Gaussian pyramid is the sequence of the expanded images $\{g'_k\}$ generated from $\{g_k\}$ by a EXPAND function:

$$g'_k= \text{EXPAND}(g_{k+1}) \tag{2.3}$$

$$g'_k(i,j) = 4\sum_{p=-2}^{2} \sum_{q=-2}^{2} w(p,q) g_{k+1}((i-p)/2, (j-q)/2) \tag{2.4}$$

where only the terms with (i-p) and (j-q) all even are considered.

### Laplacian Pyramid

The Laplacian pyramid $\{L_k\}$ is just the difference of $\{g_k\}$ and $\{g'_k\}$: with $L_{H-1}= g_{H-1}$,

$$L_k= g_k - g'_k ,\ 0 =< k < H-1. \tag{2.5}$$

## 3. PARALLEL IMPLEMENTATION

In our implementation, a multi-SIMD quad-pyramid computer consists of "$\log_2 m + 1$" layers of mesh-connected processors, where m*m is the size of the base mesh. Between meshes, there are vertical connections which form a quad-tree with the connected processors. All communication links are bi-directional and used to transfer data between processors. Inside of a processor, there are a local memory and a processing unit that can perform standard arithmetic and logic operations. With one controller per layer, all processors of a layer execute the same instructions in parallel. This pyramid scheme can be simulated on a mesh connected computer. The layer k is formed by keeping one out of $2^k*2^k$ processors in the mesh. The communications on the layer k with either the nearest neighbors or the parent take $2^k$ elementary communication steps. Both pyramid and mesh connected computers here are bit serial machines.

On bit serial machines, a multiplication of two b-bit variables takes a time of bxb elementary steps while a communication of a b-bit variable takes only b elementary steps. An efficient implementation will accordingly minimize the number of multiplications.

In this section, two cases are considered: Small images or large images compared with the pyramid machine's base. Parallel implementation and performance evaluations are presented for these two cases. Improvements obtained by exploiting the symmetry property of the Gaussian weighting function are also discussed.

## 3.1 Implementation for Small Images

Small image means that the pyramid machine has at least the same size as the Laplacian pyramid of the image. A 5-by-5 window is used for the Gaussian-like weighting function w(). The implementation is similar for other sizes of windows. We consider here the case of general weighting function without coefficient symmetries. As the window consists of 25 possibly different coefficients, a naive implementation would lead to a 25 multiplications and communications. As multiplication is the most expansive operation, we will show that this number can be largely reduced by performing several operations in parallel.

Construction of the Gaussian pyramid $\{g_k\}$

As we will see later, the implementation will be easier, if a 6-by-6 window is used. However a 6-by-6 window can be defined without changing the REDUCE function, as $\{w(p,q)\}_{[-2,+3][-2,+3]}$ where: w(p,q)=0, for p=3, or q=3; and w(p,q) remains the same as in the 5-by-5 window for other values of p and q. The formula 2.2 become:

$$g_{k+1}(i,j) = \sum_{p=-2}^{3} \sum_{q=-2}^{3} w(p,q) g_k(2i+p, 2j+q). \qquad (3.1)$$

Now, let's put $g_0$ (the original image) on the base of the pyramid machine with one pixel per processor. Assuming $g_k$ already computed and stored on the layer k as $g_0$ on the layer 0, the problem become: on the layer k+1, for each pixel (i,j), how to compute $g_{k+1}$ and to store the result in processor (i,j).

If we split the image in 2-by-2 blocks, with the previous 6-by-6 window, every block will have 9 different contributions(see Fig. 3.1). The 6-by-6 window is divided into 9 blocks as shown in Tab. 3.1. If the window center w(0,0) is put on the processor (2i,2j) of the layer k, as shown in Fig. 3.1, we can make a local sum in each block, called l_sum(b) where b is the index of block. For example:

$$l\_sum(1) = \sum_{p=0}^{1} \sum_{q=0}^{1} w(-2+p,-2+q) g_k(2(i-1)+p, 2(j-1)+q) \qquad (3.2)$$

We have: $$g_{k+1}(i,j) = \sum_{b=1}^{9} l\_sum(b) \qquad (3.3)$$

For the pixel (i,j), the computation is quite simple. Fig. 3.1 illustrates a 3-stage process: Stage 1) on the layer k, each processor multiples its $g_k()$ with the corresponding w(), then sends the result to its father on the layer k+1. Stage 2) on the layer k, we have the local sums l_sum() distributed

| | | |
|---|---|---|
| b=1 | b=2 | b=3 |
| w(-2,-2) w(-2,-1) | w(-2, 0) w(-2, 1) | w(-2, 2) w(-2, 3) |
| w(-1,-2) w(-1,-1) | w(-1, 0) w(-1, 1) | w(-1, 2) w(-1, 3) |
| b=4 | b=5 | b=6 |
| w( 0,-2) w( 0,-1) | w( 0, 0) w( 0, 1) | w( 0, 2) w( 0, 3) |
| w( 1,-2) w( 1,-1) | w( 1, 0) w( 1, 1) | w( 1, 2) w( 1, 3) |
| b=7 | b=8 | b=9 |
| w( 2,-2) w( 2,-1) | w( 2, 0) w( 2, 1) | w( 2, 2) w( 2, 3) |
| w( 3,-2) w( 3,-1) | w( 3, 0) w( 3, 1) | w( 3, 2) w( 3, 3) |

Tab. 3.1 The Gaussian weighting function w()

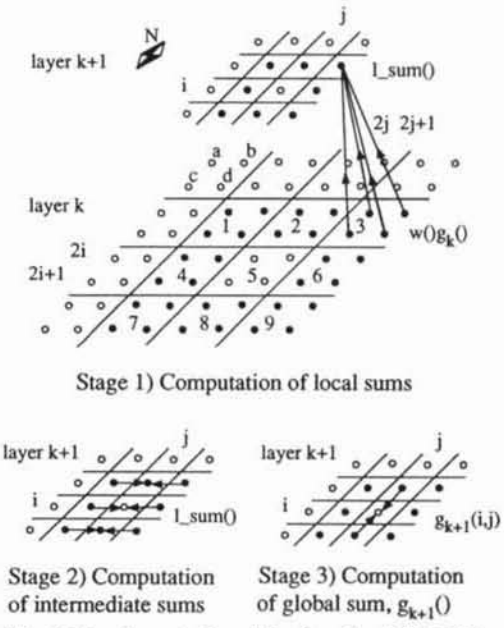
in 9 processors. By using horizontal communications, we can easily compute the intermediate sums:

$$i\_sum(r) = \sum_{b=1}^{3} l\_sum(b+3r),\ r=0..2 \qquad (3.4)$$

Stage 3) the global sum is obtain in a similar way:

$$g_{k+1}(i,j) = \sum_{r=0}^{2} l\_sum(r). \qquad (3.5)$$

For parallel computation, the layer k is divided into 2-by-2 blocks. The 4 processors in a block have the same father on the layer above. We notice that a pixel can only be at one of the 4 positions shown in Fig. 3.1, i.e. a, b, c, and d, and the block containing the pixel can be one of the 9 possible blocks in the 6-by-6 window, i.e. 1, 2, ..., 9, depending on the contributions of the pixel to compute $g_{k+1}()$. For a given pixel (i,j) on the layer k+1, the computation of its $g_{k+1}()$ is carried out by using the contributions from the processor (2i,2j) on the layer below, and its 35 nearest neighbors. So, on the layer k, each block with its 4 pixels can contribute 9 local sums l_sum() to its father. On the layer k+1, in each processor only the l_sum(5) is



layer k+1
N
j
i
l_sum()
2j 2j+1
a b
c d
layer k
1 2 3
4 5 6
7 8 9
2i
2i+1
w()g_k()
Stage 1) Computation of local sums
layer k+1
j
i
l_sum()
layer k+1
j
i
g_{k+1}(i,j)
Stage 2) Computation of intermediate sums
Stage 3) Computation of global sum, g_{k+1}()

Fig. 3.1 Implementation of the function REDUCE (Computation for the pixel (i,j))

for the computation of its own $g_{k+1}()$, others are for its neighbors. For example, l_sum(1) is for the south-east neighbor (see Fig. 3.1). By exchanging l_sum() one by one, i_sum() can be also computed in parallel. The same method is used for computing the global sum $g_{k+1}()$. Now, we have a 3-stage parallel process: Stage 1) each block on the layer k, computes in cooperation with its father, l_sum(b) where b = 1..9, one bye one. Stage 2) each processor on the layer k+1, by exchanging l_sum() with its east and west neighbors, computes i_sum(r) where r = 0..2, one by one. Stage 3) each processor on the layer k+1, by exchanging i_sum() with its south and north neighbors, computes its own $g_{k+1}()$.

In this parallel implementation, there are "9*(1 *mul* + 1 *vc* + 1 *4-add*" in the stage 1, "3*(2 *hc* + 2 *add*)" in the stage 2, and "2 *hc* + 2 *add*" in the stage 3, where "*add*", "*4-add*", "*mul*", "*hc*" and "*vc*" stand for respectively addition, 4-operand addition, multiplication, horizontal and vertical communications. The running time is independent of the layer's size. On a mesh connected computer due to the increasing horizontal or vertical communication time, the running time on the layer k is under the form of "$a+b2^k$", where "a" and "b" represent respectively computation time and communication steps.

Construction of the Expanded Gaussian pyramid $\{g'_k\}$

In the formula 2.4, only the terms with (i-p) and (j-q) all even are considered. Depending on the parity of i and j, the formula 2.4 become: (i,j all even)

$$\text{a) } g'_k(i,j) = 4\sum_{p=-1}^{1}\sum_{q=-1}^{1} w_a(p,q)g_{k+1}(i/2+p, j/2+q) \qquad (3.6)$$

$$\text{b) } g'_k(i,j+1) = 4\sum_{p=-1}^{1}\sum_{q=-1}^{1} w_b(p,q)g_{k+1}(i/2+p, j/2+q) \qquad (3.7)$$

$$\text{c) } g'_k(i+1,j) = 4\sum_{p=-1}^{1}\sum_{q=-1}^{1} w_c(p,q)g_{k+1}(i/2+p, j/2+q) \qquad (3.8)$$

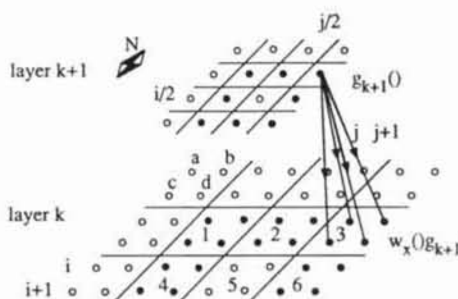
$$\text{d) } g'_k(i+1,j+1) = 4\sum_{p=-1}^{1}\sum_{q=-1}^{1} w_d(p,q)g_{k+1}(i/2+p, j/2+q) \qquad (3.9)$$

where $w_a$, $w_b$, $w_c$, and $w_d$ are defined in Tab. 3.2. In this table, x (=a,b,c,d) is the position index of the coefficient in a block as shown in Fig. 3.1.

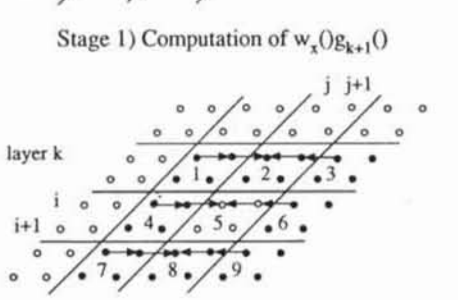
| | | | | | |
|---|---|---|---|---|---|
| b=1, $w_x(-1,-1)$ = | | b=2, $w_x(-1, 0)$ = | | b=3, $w_x(-1, 1)$ = | |
| w( 2, 2) | 0 | w( 2, 0) | w( 2, 1) | w( 2, -2) | w( 2,-1) |
| 0 | 0 | 0 | 0 | 0 | 0 |
| b=4, $w_x(0,-1)$ = | | b=5, $w_x(0, 0)$ = | | b=6, $w_x(0, 1)$ = | |
| w( 0, 2) | 0 | w( 0, 0) | w( 0, 1) | w( 0,-2) | w( 0,-1) |
| w( 1, 2) | 0 | w( 1, 0) | w( 1, 1) | w( 1,-2) | w( 1, -1) |
| b=7, $w_x(1,-1)$ = | | b=8, $w_x(1, 0)$ = | | b=9, $w_x(1, 1)$ = | |
| w(-2, 2) | 0 | w(-2, 0) | w(-2, 1) | w(-2,-2) | w(-2,-1) |
| w(-1, 2) | 0 | w(-1, 0) | w(-1, 1) | w(-1,-2) | w(-1,-1) |

Tab. 3.2 The function $w_x()$ with x=a, b, c, and d

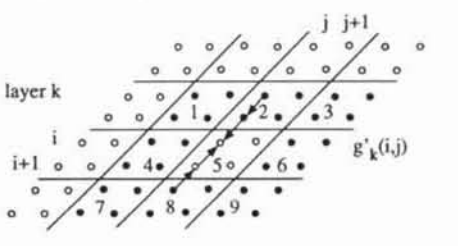
Assuming i and j all even , let's put the center of the window $w_x()$, i.e. $w_a(0,0)$, on the processor (i,j) on the layer k, as shown in Fig. 3.2. If each processor on the layer k+1 sends its $g_{k+1}()$ to its 4 children, after the formula 3.6, $g'_k(i,j)$ is just the sum of $g_{k+1}()$ stored in the processors at position "a" in the blocks from "1" to "9" multiplied by the corresponding $w_a()$. The process remains the same for $g'_k(i,j+1)$, $g'_k(i+1,j)$, and $g'_k(i+1,j+1)$. So, the parallel computation can be similar as for REDUCE, except that the data movement here is from upper layer to lower layer, and the horizontal communications are of 2 steps, i.e. the distance between two processors who exchange data is 2. A similar 3-stage process is used. Fig. 3.2 illustrates the computation of $g'_k()$ for the pixel (i,j). In the present parallel implementation, there are "1 *vc* + 9*1 *mul*" in the stage 1, "3*(4 *hc* + 2 *add*)" in the stage 2, and "4 *hc* + 2 *add* + 1 *mul*" in the stage 3. The running time is also independent of the layer's size. On a mesh connected computer, the running time remains the same form as for REDUCE.



Stage 1) Computation of $w_x()g_{k+1}()$



Stage 2) Computation of intermediate sums



Stage 3) Computation of global sum, $g'_k()$

Fig. 3.2 Implementation of the function EXPAND (Computation for the pixel (i,j))

Construction of Laplacian pyramid $\{L_k\}$

A subtraction in each processor on the layer k is sufficient to obtain $L_k = g_k - g'_k$. The running time remains independent of the layer's size. Finally, the present parallel implementation reduces the overall running time of the Laplacian pyramid algorithm to O(log n) or O(H) where H is the pyramid's height and generally less than or equal to ($\log_2 n$ + 1), and to O(n) or O($2^H$), if a mesh connected computer is used.

### 3.2 Exploitation of the Weighting Function's Symmetry

In general, the Gaussian weighting functions are separable and symmetric that lead to the following properties:

w(p,q) = w($\pm$p,$\pm$q), and w(p,q) = w(q,p).

Depending on the position of a pixel within the mesh, the different coefficients for computing the pixel's contributions are:

for (2i,2j), w(0,0), w(0, 2) and w(2,2);

for (2i+1,2j) or (2i,2j+1), w(0,1) and w(1,2);

for (2i+1,2j+1), w(1,1).

So, the maximum number of multiplications for a pixel is reduced from 9 to 3, both in REDUCE and EXPAND. In the previous 3-stage process, the number of multiplications in stage 1 is then minimized to 3. As the multiplication is the most expansive operation, the total running time is considerably reduced in this case.

### 3.3 Implementation for Large Images

When the pyramid machine is smaller than the Laplacian pyramid, we should divide the image into segments and make the computation segment by segment. However, we encounter a problem on the periphery of each segment. Indeed, the computation on the periphery of a segments needs generally the contributions from pixels of other segments nearby. To remedy this problem, we propose two solutions.

In the first solution, we use the two lowest levels of the pyramid machine to compute each layer of the Laplacian pyramid, except the layers whose size is smaller than m*m. For these layers of small size, the computation can be done by the method presented in the subsection 3.1. For the construction of the Gaussian pyramid $\{g_k\}$, we divide $g_{k+1}$ (the image to be generated) into $m_s$*$m_s$ segments and distribute $g_k$ on the base of the machine so that each segment of $g_{k+1}$ can be computed in one pass with the parallel method presented in the subsection 3.1. The $m_s$*$m_s$ segments of $g_{k+1}$ once generated, will be redistributed on the base for the computation of $g_{k+2}$. To simplify the redistribution and reduce the running time, $m_s$ should be under the form of $2^k$ where k is an integer and as bigger as possible. With a 5-by-5 weighting window, the optimal value of $m_s$ is m/4. According to the evaluation in the subsection 3.1, the running time for generating a segment is independent of the segment's size. However, the redistribution of a segment to the base of the machine needs a running time of O(m) due to the horizontal communications. So, for the layer k, the running time is proportional to $[(n*n)/(2^k*2^k)]*m/[(m/4)*(m/4)]$ or $2^{4-2k}*(n^2/m)$. The method is similar for the expanded Gaussian pyramid $\{g'_k\}$. Finally, the running time for the Laplacian pyramid's construction is $O(n^2/m + \log m)$. "log m" is for the layers whose size is smaller than m*m.

The second solution consists of simulating a big pyramid machine on a small one. Each processor on the base of the machine simulates a pyramid machine whose base's size is (n/m)*(n/m). The pyramid machine and the simulated ones form a pyramid of "$\log_2 n + 1$" levels. We use the parallel computation presented in the subsection 3.1 on this simulated pyramid machine. For the layer k, the running time is proportional to $[(n*n)/(2^k*2^k)]/(m*m)$ or $(n^2/m^2)/2^{2k}$. For the construction of the whole Laplacian pyramid, the running time is $O(n^2/m^2+\log m)$. This solution is more efficient than the first one.

### 3.4 Simulated Time Results

A quad-pyramid machine was used for the parallel implementation presented before. The method is still valid for a bin-pyramid machine like SPHINX. As the SPHINX prototype was not yet ready, the implementation was carried out on a SPHINX simulator using *Lisp (a parallel programming language used by the Connection Machine[4]). Tab. 3.3 gives some results of the running time in function of the image's size and the machine's size. All results are based on the following characteristics of SPHINX: 16 MHz clock and 300 ns for 1 bit serial instruction. The weighting function w() here is symmetric.

| n | m | H (layers) | w() (bits) | $g_k$, $L_k$ (bits) | running time (ms) |
|---|---|---|---|---|---|
| 512 | 512 | 5 | 8 | 8 | 1.2 |
| | | | | 16 | 2.4 |
| 512 | 256 | 5 | 8 | 8 | 1.9 |
| | | | | 16 | 3.8 |
| 512 | 128 | 5 | 8 | 8 | 4.5 |
| | | | | 16 | 9.0 |
| 512 | 64 | 5 | 8 | 8 | 15.7 |
| | | | | 16 | 31.4 |

Tab. 3.3 Some simulated time results

## 4. CONCLUSION

We have presented a parallel implementation of the Laplacian pyramid coding on SIMD computers. Thanks to a specific scheduling of data manipulation, the number of operations is minimized. The overall running time is reduced to O(log n) on a pyramid computer and to O(n) when the pyramid structure is simulated on a mesh. The case of large images has also been discussed. The best one of the two proposed solutions gives an overall running time of $O(n^2/m^2+\log m)$.

## REFERENCES

[1] K. E. Batcher, "Design of a massively parallel processor," IEEE Trans. Computer, Vol. 29, No. 9, pp. 836-840, Sept. 1980.

[2] P. J. Burt and E. H. Adelson, "The Laplacian pyramid as a compact image code," IEEE Trans. Commun., Vol. 31, No. 4, pp. 532-540, Apr. 1983.

[3] V. Cantoni, M. Ferretti, S. Levialdi, and R. Stefanelli, "PAPIA: Pyramidal Architecture for Parallel Image Analysis," Proc. 7th Symposium on Computer Arithmetic, Urbana IL, USA, pp. 237-242, 1985.

[4] W. D. Hillis, "The Connection Machine," The MIT Press, 1985.

[5] A. Mérigot, B. Zavidovique, and F. Devos, "SPHINX, a pyramidal approach to parallel processing," Proc. Comp. Arch. for Pat. Anal. and Image Database Management, pp. 107-111, IEEE Computer Society Press, 1985.

[6] A. N. Netravali and B. G. Haskell, "Digital pictures: representation and compression," Plenum Press, 1988.