January-Monday 17, 1994

Published in
'Applications of Wavelets
to Chemical Engineering'
eds. Basu Joseph and
Rudolph Motard, pp.275-319
1994

CHAPTER 8 : A PARALLEL TWO DIMENSIONAL WAVELET PACKET TRANSFORM AND SOME APPLICATIONS IN COMPUTING AND COMPRESSION ANALYSIS.

Marie FARGE[a1]
Eric GOIRAND[b12]
Mladen Victor WICKERHAUSER[b3]

[a]LMD-CNRS
Ecole Normale Supérieure
24 rue Lhomond
75231 PARIS Cedex 05, FRANCE

[b]Washington University
Department of Mathematics
1 Brookings Drive, Campus Box 1146
SAINT-LOUIS, MISSOURI, 63130, USA

Abstract

In a first part, we study a parallel algorithm (written for a MIMD machine) to compute the two-dimensional wavelet packet transform. Then, we apply it to compute the multiplication of a matrix by a vector in parallel. In a last part, we will use it to analyze and compress a two-dimensional turbulent field.

[1]Supported in part by NATO contract CRG-93456
[2]Supported in part by ONERA
[3]Supported in part by AFOSR and NSF

TABLE OF CONTENTS

8.1. INTRODUCTION

We will present an implementation on a MIMD computer of the two-dimensional periodic wavelet packet transform, the two-dimensional "best-basis" choice algorithm, the non-standard matrix multiplication algorithm [1] and some analysis and compression techniques for studying two-dimensional turbulent fields. Our implementation also works for the wavelet transform numerical algorithms of Beylkin, Coifman and Rokhlin [2]. It is one way to obtain a fast functional calculus for certain classes of linear operators ; those operators, which sparsify in the wavelet basis or the "best-basis" of the wavelet packets, can be applied to vectors in a lower order of complexity. The purpose of parallelizing the transform is to distribute the cost of the initial sparsification "investment" over a large number of machines. This one-time cost is $O(N^2 \log(N))$ with a non negligible constant; we envision applications in which $N=10^6$, for example evolutions of 2-dimensional fluid velocity fields on 1000x1000 point grids.

We will compute matrix coefficients for operators with respect to an orthonormal basis of separable wavelet packets, using the 2-dimensional version of the fast wavelet packet transform [3], [4]. The main idea is to lift an NxN matrix, which maps R^N to R^N , into the space of maps from $R^{\{N\cdot\log(N)\}}$ to $R^{\{N\cdot\log(N)\}}$. Any of a large number of NxN coefficient subsets of this lifting can be used to represent the operator, so we may pick the subset in which the matrix is most sparse. The choice is made with the "best-basis" algorithm and is itself a fast algorithm. The number of basis subsets of this type grows like 4^N for an NxN matrix, but computing all of the coefficients with respect to all of the basis functions requires just $O(N^2 \log(N))$ operations. The algorithm for choosing the "best-basis" in which the operator appears most sparse is described in [5]; it has a complexity of $O(N^2)$. We have reason to believe that the equations of motion for large physical systems sparsify by nearly an order of magnitude in this collection of bases. We thus obtain lower complexity matrix application and matrix multiplication algorithms from the new representation. Of course the method is not perfectly general : the speedup depends very much upon the problem. However, others have shown that for broad classes of operators we can expect an order of magnitude asymptotic complexity reduction for matrix multiplication.

As an application, we will use the wavelet packet representation of a field to extract parts of it, either by keeping only its significative coefficients or by selecting a particular zone. In this new representation of the field, we can then perform some well-known analyzes , to get some local information. As an example of this method, we will in the last part of this chapter analyze and compress a two-dimensional turbulent field.

8.2. PARALLEL WAVELET PACKET DECOMPOSITION

Suppose we have a two-dimensional periodic signal S = $(s_{i,j})$ with $0 \le i < 2^{Xmax}$ and $0 \le j < 2^{Ymax}$, and suppose nmax = min(Xmax,Ymax). We want to calculate all the wavelet packets ${}^{f_j}C^j$ or rather the sets of wavelet packet coefficients ${}^{f_j}C^j_{k_j,l_j}$ defined as follows :

- The indices are in the range :

$$\left.\begin{array}{l} 0 \le j \le nmax \\ 0 \le k_j < 2^{Xmax-j} \\ 0 \le l_j < 2^{Ymax-j} \\ 0 \le f_j < 4^j \end{array}\right\} \quad with\ j, k_j, l_j\ and\ f_j \in \mathrm{N}$$

- The initial packet is :

$${}^0C^0_{k,l} = s_{k,l}\ with\ 0 \le k < 2^{Xmax}, 0 \le l < 2^{Ymax}$$

- Then every other packet is defined recursively by the formula :

$${}^{f_{j+1}=4.f_j+d}C^{j+1}_{k_{j+1},l_{j+1}} = \sum_{n_j,m_j} F(d,f_j)_{n_j-2k_{j+1},m_j-2l_{j+1}} * {}^{f_j}C^j_{n_j,m_j} \quad \text{with}$$

d = 0, 1, 2 or 3. We shall say that the packet ${}^{f_j}C^j$ is the father of the four packets ${}^{4\cdot f_j+d}C^{j+1}$ or that they are its four children.

- From a one-dimensional wavelet Ψ (defined by its filter G) and its smoothing function Φ (defined by its filter H) we obtain four two-dimensional filters by tensor products :

$$\begin{cases} Filter(0)_{n,m} = H_n \cdot H_m \\ Filter(1)_{n,m} = H_n \cdot G_m \\ Filter(2)_{n,m} = G_n \cdot H_m \\ Filter(3)_{n,m} = G_n \cdot G_m \end{cases}$$

In order to keep the frequencies (along the x- and y-axis) increasing, we then define $F(d,f_j)_{n,m} = Filter(gray\_code2d(d,f_j))_{n,m}$ where :

$$gray\_code2d(d,f_j) = 2 \cdot a + b$$

$$a = \begin{cases} d \bmod 2 & if\ f_j\ is\ even \\ 1-(d \bmod 2) & if\ f_j\ is\ odd \end{cases}$$

$$b=\begin{cases}\left[\dfrac{d}{2}\right] & \text{if } \left[\dfrac{f_j}{2}\right] \text{ is even}\\ 1-\left[\dfrac{d}{2}\right] & \text{if } \left[\dfrac{f_j}{2}\right] \text{ is odd}\end{cases}$$

The symbol [x] is the integer part of x.

- At a scale j+1, there are four times as many wavelet packets as there are at scale j, and each wavelet packet has four times fewer coefficients than those at scale j.
- From the first remark, with an initial signal of 2^{Xmax} by 2^{Ymax} points, we deduce that :
  . At scale j, there are 4^j wavelet packets, each with 2^{Xmax-j} by 2^{Ymax-j} coefficients, so we still have $2^{Xmax} \times 2^{Ymax}$ coefficients.
  . We can then use the following representation of the two-dimensional wavelet packet decomposition.

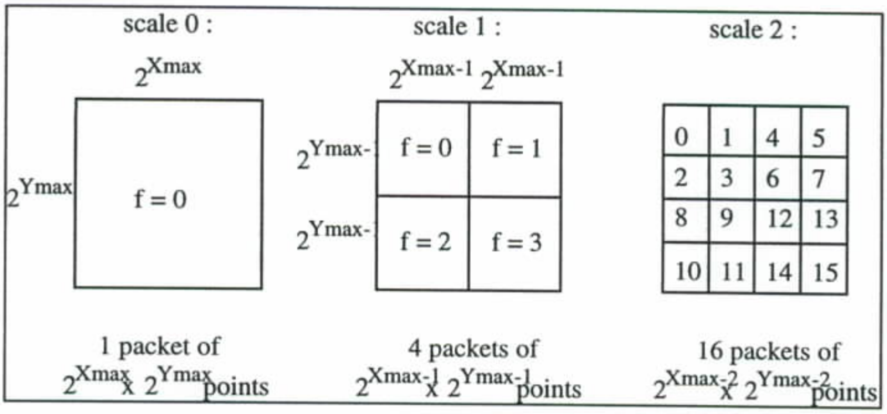

scale 0 :
2^{Xmax}
2^{Ymax}
f = 0
1 packet of $2^{Xmax} \times 2^{Ymax}$ points
scale 1 :
2^{Xmax-1} 2^{Xmax-1}
2^{Ymax-1}
2^{Ymax-1}
f = 0
f = 1
f = 2
f = 3
4 packets of $2^{Xmax-1} \times 2^{Ymax-1}$ points
scale 2 :
0 1 4 5
2 3 6 7
8 9 12 13
10 11 14 15
16 packets of $2^{Xmax-2} \times 2^{Ymax-2}$ points

Figure 8.2.1 : Data arrangement for a 2D wavelet packet decomposition

  . If Xmax=Ymax=nmax, then at scale nmax, we have 4^{nmax} wavelet packets of 1 coefficient each.

To do the decomposition in parallel, we will separate each scale (represented by a field of 2^{Xmax} by 2^{Ymax} points) into 4^{jp} $(0 \leq jp < nmax)$ parts of equal size. Each process always stores the same part of the field so we can identify the process name by the name of the packet it contains at scale jp. For example, dividing the wavelet packet onto 4 processes (jp = 1) from scale 0 to 2 gives the following :

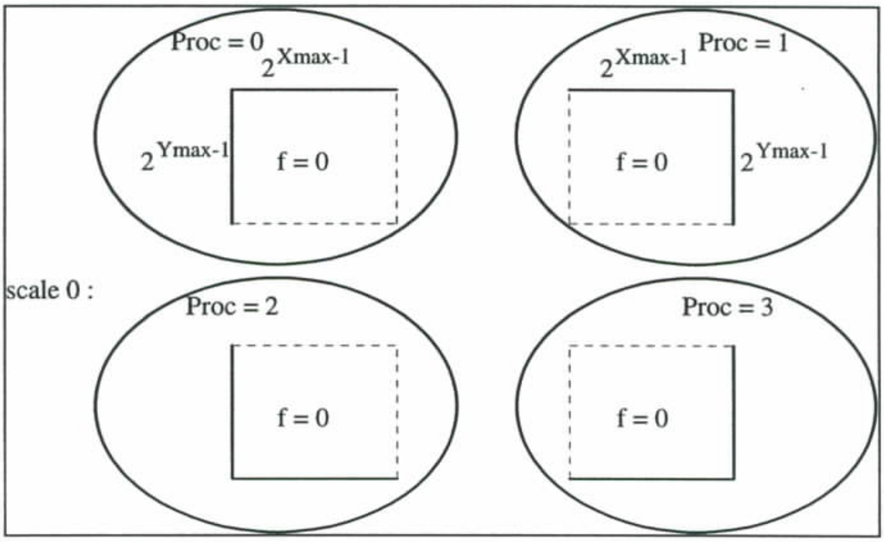


Figure 8.2.2 :
Repartition of the wavelet packets of scale 0 onto 4 processes

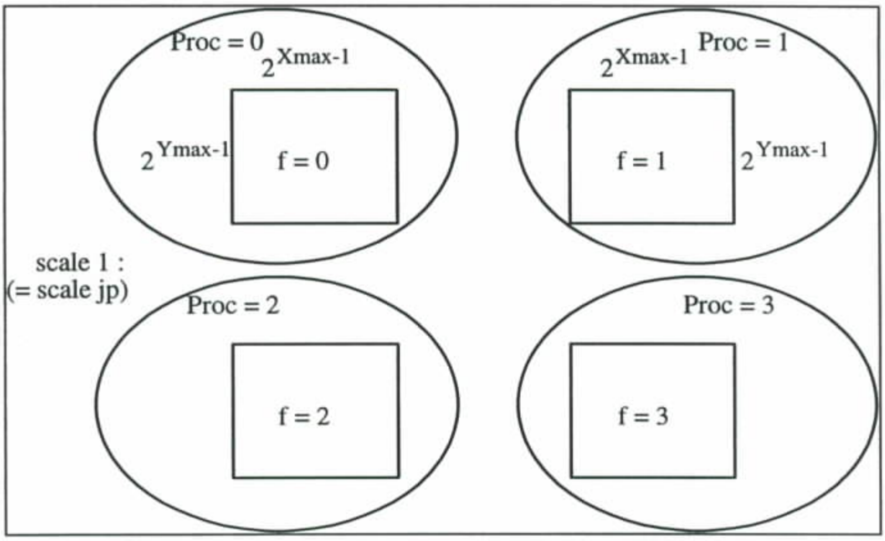


Figure 8.2.3 :
Repartition of the wavelet packets of scale 1 onto 4 processes

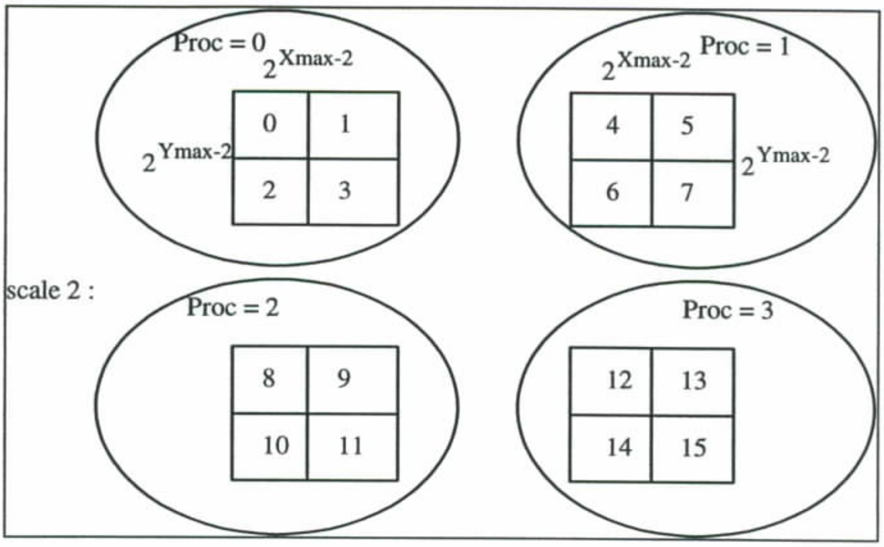


Figure 8.2.4 :
Repartition of the wavelet packets of scale 2 onto 4 processes

From the decomposition formula, one sees how the packets at scale j+1 are obtained from those at scale j. To calculate the decomposition in parallel, after cutting up the field as shown above, we will use the formula in two different ways depending upon whether j is smaller or greater than jp.

First, let us consider the case when j < jp.
A packet at scale j is shared by 4^{jp-j} processes. In the initial step, those processes will exchange their data in order to have the whole shared packet on each of them. Then, in the second step, they will calculate their own part of the packet they share at scale j+1. This part will be the entire child packet when scale j+1 equals jp.

From the decomposition formula we obtain that at scale j, the frequency f of the packet stored on process p is given by $f=\left[\frac{p}{4^{jp-j}}\right]$ where [x] is the integer part of x. Reciprocally, a packet at scale j and frequency f is shared by the processes $p=4^{jp-j}\cdot f$ to $p=4^{jp-j}\cdot(f+1)-1$. We shall say that the first one is the base process and all processes sharing the same packet are companions. The base process is the one with the upper left-hand corner of the wavelet packet array.

In the same way, at scale j, the position (base_x, base_y) of the upper left corner of the packet stored on process p is given by :

- $base\_x = cx\left(p,\ jp,\ 2^{Xmax}\right) - cx\left(\left[\frac{p}{4^{jp-j}}\right], j,\ 2^{Xmax}\right)$ where

$$cx(p, jp, 2^{Xmax}) = \begin{cases} 0 & \text{if } jp=0 \\ \sum_{m=1}^{jp} 2^{Xmax-m}.\left\{\left[\frac{p}{4^{jp-m}}\right] \bmod 2\right\} & \text{otherwise} \end{cases}$$

- $base\_y = cy\left(p,\ jp,\ 2^{Ymax}\right) - cy\left(\left[\frac{p}{4^{jp-j}}\right], j,\ 2^{Ymax}\right)$ where

$$cy(p, jp, 2^{Ymax}) = \begin{cases} 0 & \text{if } jp=0 \\ \sum_{m=1}^{jp} 2^{Ymax-m}.\left\{\left[\frac{\left[\frac{p}{4^{jp-m}}\right]}{2}\right] \bmod 2\right\} & \text{otherwise} \end{cases}$$

The width and height of the part of the packet stored on a process at scale j is $2^{Xmax-jp}$ by $2^{Ymax-jp}$.

Example : Suppose we have 16 processes (jp=2) and we try to calculate which processes share the packet at scale j=1 and frequency f=2.

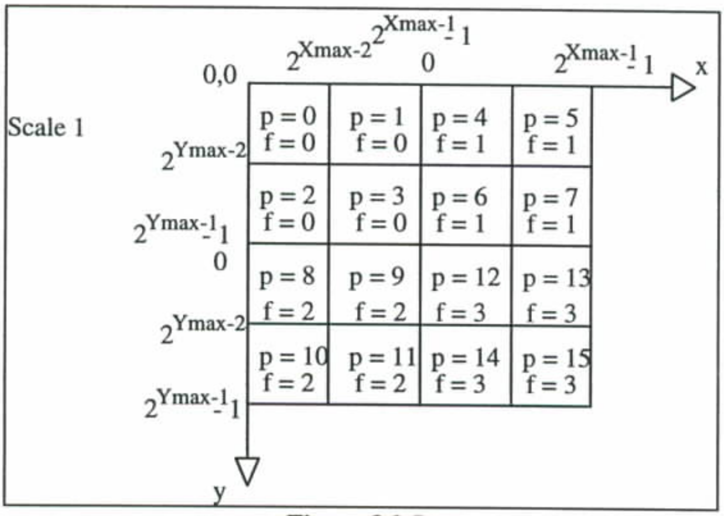


Figure 8.2.5 :
Example of the repartition of wavelet packet coefficients at scale 1 with 16 processes.

From these hypotheses, $f = \left[\frac{p}{4}\right] = 2$, so we deduce from the former formulas that the processes sharing ${}^{2}C^{1}$ are numbered *8* to *11*. Each of them has wh = $2^{\text{Xmax-jp}}$ by ht = $2^{\text{Ymax-jp}}$ data points arranged as follows :

Process *8* contains the local packet ${}^{2}lC^{1}_{0..wh-1,\,0..ht-1} = {}^{2}C^{1}_{0..wh,0..ht-1}$.

Process *9* contains the local packet ${}^{2}lC^{1}_{0..wh-1,\,0..ht-1} = {}^{2}C^{1}_{wh..2wh-1,0..ht-1}$

Process *10* contains the local packet ${}^{2}lC^{1}_{0..wh-1,\,0..ht-1} = {}^{2}C^{1}_{0..wh,ht..2ht-1}$

Process *11* contains the local packet ${}^{2}lC^{1}_{0..wh-1,\,0..ht-1} = {}^{2}C^{1}_{wh..2wh-1,ht..2ht-1}$

We will do the transfers as follows, looping in the index jj :

- We start with jj equals jp-1 and define a pivot process named depl, where $depl = 4\cdot\left[\frac{p}{4^{jp-jj}}\right]$. In the initial step, processes *p* verifying $\left[\frac{p}{4^{jp-jj}}\right] = depl+i$ (A) exchange their data with processes $\left[\frac{p}{4^{jp-jj}}\right] = depl+1+i$ (B) where i equals 0 or 2. At the end of this step, the processes (A) and (B) share the same data. In the second step, processes verifying $\left[\frac{p}{4^{jp-jj}}\right] = depl+i$ (C) and processes verifying $\left[\frac{p}{4^{jp-jj}}\right] = depl+2+i$ (D) with i equals 0 or 1 will exchange the data they got from step one. At the end of these two steps, the four processes share the same data.
- We decrement jj by one and repeat until jj equals j.
- At this point, each process sharing a packet contains all the coefficients of that packet.

For definiteness, let us consider an example :

Suppose we have four processes (jp = 1) and we want to calculate the packets at scale 1 from the packet at scale 0. We have j=0, jj = jp-1=0. In this example, there will be only one loop over jj. In the first step process 0 exchanges its data with process 1 (while process 2 exchanges its data with process 3). Then in the second step process 0 exchanges its new set of data (its initial data plus process 1 initial data) with process 2 new set of data (while process 1 does the same with process 3). We exit the loop on jj, and every processes now carries the packet ${}^{0}C^{0}$.

The pseudo-code representation of the algorithms described above is given in the annex 1 "Decomposition".

To calculate the transfer complexity, we will assume that the cost of a data transfer is independent of the size of the data set. This is approximately true for a large size data set. Thus, the transfer complexity will only show an order of the number of transfers without any specification of the sizes of the data sets.

Counting passes through the jj loop, we see that the transfer complexity for the function *Data transfers* is then 4*(jp-j).

In this previous algorithm (when scale $j < jp$), the function *Data transfers* is done jp times so we deduce that the total transfer complexity is of order O(2*jp*(jp+1)). The decomposition complexity is of order $O(jp*2^{Xmax}*2^{Ymax}/4^{jp})$, where the constant depends on the length of the filters. In simple terms, if $NP=4^{jp}$ is the number of processes and if we have $N=4^{nmax}$ points (Xmax=Ymax=nmax), then the transfer complexity is $O(2*[\log_4(NP)]^2)$ and the decomposition complexity is $O(P*\log_4(NP)*N/NP)$.

Second, let us consider the case when $j \geq jp$.
Each process has 4^{j-jp} packets, and from each of these, we need to calculate their four children. In this case, there is no parallelism as the children also belong to the process. We will then use the sequential algorithm on the subset of packets each process carries at scale j.

For $j \geq jp$, there is no transfer complexity. The decomposition complexity is of order $O((nmax-jp)*2^{Xmax}*2^{Ymax}/4^{jp})$. If we consider $NP = 4^{jp}$ processes and $N = 4^{nmax}$ points, then the total complexity of this part is $O((\log_4(N)-\log_4(NP))*N/NP)$.

Combining the calculations from the two different parts of the parallel decomposition algorithm, we conclude that the transfer complexity is of order O(2*jp*(jp+1)) and the computational complexity is of order $O(nmax*2^{Xmax}*2^{Ymax}/4^{jp})$.

Thus, if $N=4^{nmax}$ is the number of points (Xmax=Ymax=nmax) and $NP=4^{jp}$ is the number of processes then the transfer complexity is $O(2*[\log_4(NP)]^2)$ and the computational complexity is $O(\log_4(N)*N/NP)$.

8.3. PARALLEL WAVELET PACKET BEST BASIS SELECTION

At the end of the decomposition, we have obtained the complete discrete analysis of the input data. We must now extract a subset of wavelet packets forming a basis. There are many methods to extract a basis : keeping only the wavelet packets at a certain scale, for example, gives one solution.

But if we consider a family of wavelet packets ${}^{f}C^{j}$ *with* $j \in J$ *and* $f \in F_J$, selecting a basis among all those packets means that if a packet ${}^{f_0}C^{j_0}$ belongs to the basis, all its descendants, the packets ${}^{4*f_0+d_0}C^{j_0+1}$, ${}^{4*(4*f_0+d_0)+d_1}C^{j_0+2}$, etc ... (with d_0, d_1 equals 0,1,2 or 3) cannot belong to the same basis.

We will maximize a concentration criterion to find a "best basis". For each packet ${}^{f}C^{j}$, we compute a value $M\left({}^{f}C^{j}\right)$ using a cost function M on the set.

For example, we can take a measure M which counts the non null wavelet packet coefficients of a wavelet packet. Another commonly used measure is Me, which evaluates the entropy of a wavelet packet expansion :

$$\left\{\begin{array}{l} Me(\varnothing) \;=\; 0 \\ Me({}^{f}C^{j}) = -\sum\limits_{k,l}({}^{f}C^{j}_{k,l})^2 * \ln\left[({}^{f}C^{j}_{k,l})^2\right] \end{array}\right\}.$$

Once the measure is chosen, we shall produce a family of wavelet packets that satisfies the condition for being a basis and that minimizes the value of the measure.

This is done by an iterative algorithm. We define ${}^{f}C^{j}_{value}$ to be the value of the measure of the best basis of wavelet packets in the subset below ${}^{f}C^{j}$. First, for all the frequencies f at scale j=nmax, we initialize ${}^{f}C^{j}_{value} = M\left({}^{f}C^{j}\right)$. Then we start the algorithm at scale j=nmax-1. For all the frequencies f at scale j, we compare $v_0 = M\left({}^{f}C^{j}\right)$ to $v_1 = \sum\limits_{d=0}^{3} {}^{4*f+d}C^{j+1}_{value}$. If v_0 is smaller than or equal to v_1 we keep the wavelet packet ${}^{f}C^{j}$, we set ${}^{f}C^{j}_{value} = v_0$, and we discard all its descendants. If v_0 is greater than v_1, we keep the four children packets ${}^{4*f+d}C^{j+1}$, and we set ${}^{f}C^{j}_{value}$ to v_1. We decrement j by one and iterate until j equals 0. At scale 0, all the kept packets form the "best basis" of the signal.

To select the best basis in parallel, we will use how the wavelet packets are stored on the different processes after the decomposition. Thus, if we have 4^{jp} processes, then from scale 0 to scale jp-1 the wavelet packets will be shared by more than one process, while from scale jp to nmax they will be part of a single process.

From the definition of the best basis selection, one can see that there will be two different algorithms, depending on whether j is greater or smaller than jp. In order to know if a packet is kept or not, we define a variable ${}^{f}C^{j}_{status}$ (${}^{f}lC^{j}_{status}$

for a local packet) equal to *KEPT* if the wavelet packet ${}^{f}C^{j}$ is part of the best basis, and *NOT_KEPT* if it is not.

Before selecting the best basis, we will suppose that the user has already chosen a measure M and has already computed the value $M\left({}^{f}C^{j}\right)$ for every packet of the decomposition. Hence it will not be counted in the computational complexity.

First, let us consider the case when j ≥ jp.
In that case, the best basis algorithm is the same as the sequential one except that it is done on each subset of wavelet packets which belongs to each process.

The computational complexity of this algorithm is of order O(4/3*($4^{nmax-jp}$ - 1)). Therefore, if we have NP=4^{jp} processes and N=4^{nmax} points, this complexity becomes O(4/3*(N/NP - 1)).

Second, let us consider the case when j < jp.
From the parallel decomposition, we have seen that at scale j, process p contains the local packet ${}^{\left[\frac{p}{4^{jp-j}}\right]}lC^{j}$. At scale j, the packet ${}^{f}C^{j}$ is then shared by the processes $p=4^{jp-j}\cdot f$ to $p=4^{jp-j}\cdot(f+1)-1$. With another small calculation, one can see that the packets ${}^{4*f+d}C^{j+1}$ $(d=0..3)$ are shared by this same list of processes.

From these three remarks, we deduce that if we add up $lv_0 = M\left({}^{\left[\frac{p}{4^{jp-j}}\right]}lC^{j}\right)$ for all the processes $p=4^{jp-j}\cdot f\_base$ to $p=4^{jp-j}\cdot(f\_base+1)-1$ where $f\_base=\left[\frac{p}{4^{jp-j}}\right]$, we will obtain $v_0 = M\left({}^{f\_base}C^{j}\right)$. Similarly, if we add up ${}^{\left[\frac{p}{4^{jp-j-1}}\right]}lC^{j+1}_{value}$ for the same processes, we will get $v_1 = \sum_{d=0}^{3} {}^{4*f\_base+d}C^{j+1}_{value}$.

One solution consists in adding up all those local values in order to get v_0 and v_1. However, this is redundant : we do the same calculation at scale j+1 and at scale j, once for calculating v_0 and the second time for v_1. But we can easily change the sequential algorithm to improve this if at each scale j, after having calculated v_0 and v_1, we set the local value ${}^{\left[\frac{p}{4^{jp-j}}\right]}lC^{j}_{value}$ to the winner.

Then, each process sharing ${}^{4\cdot f\_base+d}C^{j+1}$ will have the same local value ${}^{4*f\_base+d}lC^{j+1}_{value}$, so the calculation of v_1 as we have already explained it, does not give v_1 but gives $4^{jp-j-1}\cdot v_1$. Although it seems worse than the first solution, this approach is in fact better, because the calculation of v_1 needs only the addition of four local values instead of 4^{jp-j}. Indeed, we can now calculate v_1 by the following formula : $v_1=\sum_{i=0}^{i=3}{}^{\left\lfloor\frac{depl+\left(p \bmod 4^{jp-j-1}\right)+4^{jp-j-1}\cdot i}{4^{jp-j-1}}\right\rfloor}lC^{j+1}_{value}$ with $depl=4^{jp-j}\cdot f\_base.$

An example might clarify this formula :

Suppose that jp=2, j=0 and p=5. To calculate v_1, the previous formula gives $v_1={}^{1}lC^{j+1}_{value}+{}^{5}lC^{j+1}_{value}+{}^{9}lC^{j+1}_{value}+{}^{13}lC^{j+1}_{value}$ so process 5 only has to exchange its data with processes 1, 9 and 13.

If we store the information costs locally in this way, then the parallel best-basis search algorithm for process p at levels j < jp becomes the following :

- We start at j = jp-1.
- First, we calculate $f\_base=\left\lfloor\frac{p}{4^{jp-j}}\right\rfloor$ and $depl=4^{jp-j}\cdot f\_base.$

  Second, we calculate $v_0=\sum_{i=0}^{4^{jp-j}-1}M\left({}^{\left\lfloor\frac{depl+i}{4^{jp-j}}\right\rfloor}lC^{j}\right)$ by adding all the values of the local packets at scale j. Third, we calculate $v_1=\sum_{i=0}^{i=3}{}^{\left\lfloor\frac{depl+\left(p \bmod 4^{jp-j-1}\right)+4^{jp-j-1}\cdot i}{4^{jp-j-1}}\right\rfloor}lC^{j+1}_{value}$ by adding all the values of the local packets at scale j+1. Finally, we compare v_0 to v_1. If v_0 is smaller than or equal to v_1 then we keep the local wavelet packet ${}^{\left\lfloor\frac{p}{4^{jp-j}}\right\rfloor}lC^{j}$, we set ${}^{\left\lfloor\frac{p}{4^{jp-j}}\right\rfloor}lC^{j}_{value}$ to v_0, and we discard all its descendants. If v_0 is greater than v_1, we keep the local packet ${}^{\left\lfloor\frac{p}{4^{jp-j-1}}\right\rfloor}lC^{j+1}$, and we set ${}^{\left\lfloor\frac{p}{4^{jp-j}}\right\rfloor}lC^{j}_{value}$ to v_1.
- We decrement j by one and iterate until j equals 0.

One way to exchange the local values is to take all the processes involved and perform pairwise exchanges until every process has the total sum. Figure 8.3.1 depicts this algorithm graphically.

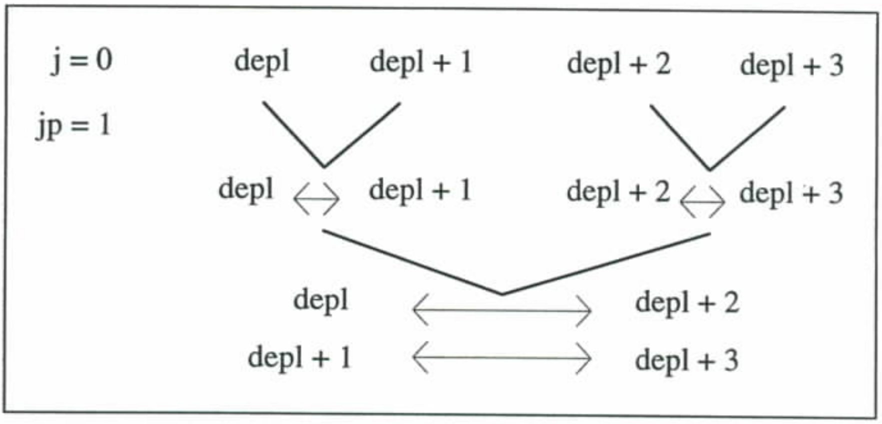


Figure 8.3.1 :
Transfer of the local values v_0 between processes depl to depl+3

To calculate v_1, we will use the same principle, except that the first transfer may not be anymore between process depl and depl+1, but might instead be between depl and depl+4^{jp-j-1}. The transfer complexity of this function is of order $O(2*\log_2(np/p\_shift))$ and the computation complexity is of order $O(\log_2(np/p\_shift))$.

The pseudo-code representation of the algorithms described above is given in the annex 2 "Best-basis".

Since we do the external loop jp times and we call the function *Local value transfer* twice, the first time with *np/p_shift* equaling 4^{jp-j} and the second time with *np/p_shift* equaling 4, we deduce that the transfer complexity is of order $O(2*jp*(jp+1))$ and the computational complexity is half of that, or $O(jp*(jp+1))$. If $NP=4^{jp}$ is the number of processes, then the transfer complexity is of order $O(2*[\log_4(NP)]^2)$ and the computational complexity is of order $O([\log_4(NP)]^2)$.

For the whole parallel best basis algorithm, the transfer complexity is of order $O(2*jp*(jp+1))$ and the computational complexity is of order $O(4/3*(4^{nmax-jp}-1) + jp*(jp+1))$.

In the case when we have $N=4^{nmax}$ points and $NP=4^{jp}$ processes, the transfer complexity is $O(2*[\log_4(NP)]^2)$ and the computational complexity is $O(4/3*N/NP + [\log_4(NP)]^2)$.

8.4. PARALLEL WAVELET PACKET RECONSTRUCTION

The reconstruction is the opposite of the decomposition, from a family of kept wavelet packets ${}^{f_j}C^j$ with $j \in J$ (where $J \subset \mathbb{N}_{nmax} = \{n \in \mathbb{N}; 0 \leq n \leq nmax\}$) and $f_j \in F_J$ (where for each $j \in J$, $F_j \subset \mathbb{N}_{4^j-1}$), we want to obtain $s_{k,l} = {}^0C^0_{k,l}$ (where $0 \leq k < 2^{Xmax}$ and $0 \leq l < 2^{Ymax}$) which is the corresponding signal representation (S). We obtain S by applying the following recursive formula until we get to ${}^0C^0$:

$$^{f_{j-1}=\left[\frac{f_j+d}{4}\right]}C^{j-1}_{k_{j-1},l_{j-1}} = \sum_{d=0}^{3} \sum_{n_j,m_j} F\left((f_j+d) \bmod 4, \left[\frac{f_j+d}{4}\right]\right)_{k_{j-1}-2n_j,\, l_{j-1}-2m_j} * {}^{f_j+d}C^j_{n_j,m_j}$$

where $F(d,f_j)_{n,m} = Filter(gray\_code2d(d,f_j))_{n,m}$ is exactly the same as for the decomposition.

Usually we do not program this formula directly, rather we reconstruct a whole branch of the tree, one level at a time. Instead of having its entire father ${}^{\left[\frac{f_j}{4}\right]}C^{j-1}$, we only have one part of it, but if we reconstruct from all the packets at scale j, we will obtain their entire fathers. The formula we program is then :

$$^{f_{j-1}=\left[\frac{f_j}{4}\right]}C^{j-1}_{k_{j-1},l_{j-1}} = {}^{f_{j-1}}C^{j-1}_{k_{j-1},l_{j-1}} + \sum_{n_j,m_j} F\left(f_j \bmod 4, \left[\frac{f_j}{4}\right]\right)_{k_{j-1}-2n_j,\, l_{j-1}-2m_j} * {}^{f_j}C^j_{n_j,m_j}$$

and the first time we encounter the packet ${}^{f_{j-1}}C^{j-1}_{k_{j-1},l_{j-1}}$ we must initialize it to the null packet (its coefficients being all zeros) if it is not a kept packet.

We suppose again that we have 4^{jp} processes, and that the data are dispatched to each of them as before. In the same way as the sequential algorithm, we will start at scale nmax and reconstruct the ancestors of each kept packet until we reach scale 0. There will again be two different algorithms, one when scale j is greater than jp, and the other when scale j is less than or equal to jp.

First, we consider the case when j > jp, the algorithm is the same as the sequential one except that it is done on the subset of wavelet packets that belong to a process.

In the worst case, the best basis is composed of all the packets at scale nmax. In that case, the computational complexity is of order $O((nmax-jp) * 2^{Xmax} * 2^{Ymax}/4^{jp})$. In all other cases it is less than this.
If we have $N=4^{nmax}$ points and $NP=4^{jp}$ processes, the computational complexity is therefore smaller than or equal to $O((\log_4(N)-\log_4(NP))*N/NP)$.

Second, we consider the case when scale j ≤ jp, we suppose initially that each process has an entire data set. Each of them will do the reconstruction locally using the sequential algorithm, and when they reach scale 0, each of them will have a partial signal. We will then need to add all these partial signals in order to get the whole signal. Unfortunately, even if we do not have a lot of transfers, the calculation complexity is exactly the same as for the sequential algorithm, so the only advantage is that we need less memory. Instead we will propose another solution which has a lower computational complexity but might require a larger number of transfers.

We proceed in the following way. At scale j (1 ≤ j ≤ jp), we reconstruct the local packet ${}^{f=\left[\frac{p}{4^{jp-j}}\right]}lC^{j}$ in a full size temporary father packet ${}^{f/4}tC^{j-1}$ using the formula : ${}^{f/4}tC^{j-1}_{k,l} = {}^{f/4}tC^{j-1}_{k,l} + \sum_{n,m} FF_{n-2(base\_x+k),m-2*(base\_y+l)} * {}^{f}lC^{j}_{n,m}$ where FF is the filter F(f mod 4,f / 4), and base_x and base_y are the x- and y-offsets of the local packet ${}^{f}lC^{j}$. At this point we must add the local packet ${}^{f/4}lC^{j-1}$ to this temporary packet if it is a kept packet. Then, each process sharing the packet ${}^{f/4}C^{j-1}$ exchanges and adds all the temporary packets in order to have the whole one. After the transfer, they keep their own part of this father ${}^{f/4}lC^{j-1}$. Then we decrement j by one and iterate until j equals 0.

One problem occurs when all concerned processes have to exchange the temporary packets : what happens when not all the processes are waiting for the transfer ? This case is not impossible, it happens when there is no kept packet when scale j is greater than or equal to j0 (with 0 ≤ j0 ≤ jp) on at least one process. We handle this problem in a very simple manner : if the packet ${}^{p}lC^{jp}$ does not exist, we pretend it is the null packet. The propagation of this packet to all scales j smaller than jp will handle the problem once and for all.

We first discuss the function which transfers packets between the concerned processes, then we explain the computational part of the reconstruction algorithm.

The function doing these transfers is similar to the local value transfer function we have presented in the best basis selection algorithm. Here, we consider the transfer and the addition of local arrays instead of local values.

The pseudo-code representation of the algorithms described above is given in the annex 3 "Reconstruction".

The transfer complexity of this function is of order O(2*log$_2$(np/p_shift)) and the computational complexity is of order O(size_array*log$_2$(np/p_shift)).

As the function *Local array transfer* is done jp times, we deduce that the transfer complexity of the algorithm is of order O(2*jp*(jp+1)) and the computational complexity is of order O(jp*(jp+1)*2^{Xmax}*2^{Ymax}/4^{jp}).
If N=4^{nmax} points and NP=4^{jp} processes, the transfer complexity is O(2*[$\log_4$(NP)]^2) and the computational complexity is O([$\log_4$(NP)]^2*N/NP).

In the worst case, if we consider that the best basis is composed of all the wavelet packets at scale nmax, the transfer complexity is O(2*jp*(jp+1)) and the computational complexity is O((nmax+jp^2)*2^{Xmax}*2^{Ymax}/4^{jp}).

If we now suppose that N=4^{nmax} points and NP=4^{jp} processes, the transfer complexity is smaller than or equal to O(2*[$\log_4$(NP)]^2) and the computational complexity is smaller than or equal to O(($\log_4$(N)+[$\log_4$(NP)]^2)*N/NP).

8.5. PARALLEL MATRIX-VECTOR MULTIPLICATION IN THE WAVELET PACKET BASIS EXPANSION

We wish to compute the product of a matrix C = ($c_{i,j}$) by a vector D = (d_i) in the wavelet packet expansion. We suppose that $0 \le i < 2^{Xmax}$ and $0 \le j < 2^{Ymax}$ so the result is a vector E = (e_j). We perform the multiplication in 3 steps.

During the first step we decompose the matrix C in a wavelet packet best basis, we obtain a family of kept wavelet packets ${}^{cf_j}C^j$ with $j \in J$ (where $J \subset \mathrm{N}_{nmax} = \{n \in \mathrm{N}; 0 \le n \le nmax\}$) and $cf_j \in F_J$ (where for each $j \in J$, $F_j \subset \mathrm{N}_{4^j - 1}$). In the second step, we decompose the vector D in all the possible one dimensional wavelet packets ${}^{df_j}D^j$ with $0 \le j \le nmax$ and $0 \le df_j < 2^j$.

Each frequency cf_j is the combinaison of a frequency cf_x_j along the x-axis and a frequency cf_y_j among the y-axis. We write that $cf_j = (cf\_x_j, cf\_y_j)$ and we calculate cf_x_j and cf_y_j by :

$$cf\_x_j = \sum_{jj=0}^{j-1} 2^{j-jj-1} . \left(\left\lfloor \frac{cf_j}{4^{j-jj-1}} \right\rfloor \bmod 2 \right)$$

$$cf\_y_j = \sum_{jj=0}^{j-1} 2^{j-jj-1} . \left(\left\lfloor \frac{\left\lfloor \frac{cf_j}{4^{j-jj-1}} \right\rfloor}{2} \right\rfloor \bmod 2 \right)$$

The multiplication in the wavelet packet expansion consists in obtaining all the packets ${}^{cf\_y_j}E^j$ which are the solution of the multiplication of ${}^{cf_j=(cf\_x_j,cf\_y_j)}C^j$ by ${}^{cf\_x_j}D^j$.

The third and final step is to reconstruct all the packets ${}^{cf\_y_j}E^j$ to obtain the vector E.

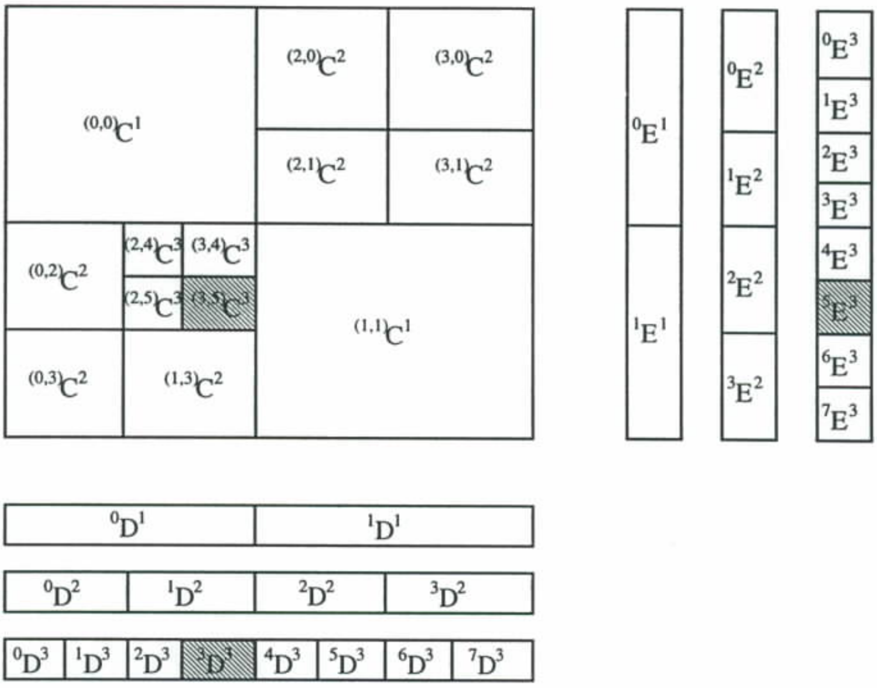


Figure 8.5.1 :
Example of the multiplication in the wavelet packet expansion,
we consider ${}^{5}E^{3} = {}^{(3,5)}C^{3} \times {}^{3}D^{3}$

We can use the parallel algorithms previously described in order to perform the first step and the multiplication.

At the beginning, we use 4^{jp} processes to decompose the matrix C in all the wavelet packets and we select its best basis. On each process, we decompose the vector D in all the one dimensional wavelet packets.

The multiplication between the wavelet packets can now be done as follows :

- for each kept packet ${}^{f}C^{j}$ at scale j, we first find the frequencies fx and fy corresponding to f, then

- if j is smaller than jp we do a multiplication between ${}^{f}lC^{j}$ and the corresponding local packet of ${}^{fx}D^{j}$ (named ${}^{fx}lD^{j}$) to obtain the local packet ${}^{fy}lE^{j}$. We insert it in the null packet ${}^{fy}E^{j}$.
- if j is greater than or equal to jp, we do the multiplication between ${}^{f}C^{j}$ and ${}^{fx}D^{j}$ to have ${}^{fy}E^{j}$.

At the end of the multiplication, each process has a partial vector E in a wavelet packet expansion (it is not usually a basis). They will then reconstruct all the kept packets to get their partial vector E.

In a last part, each process exchanges and adds all the partial vectors E in order to get the complete result of the multiplication of C by D.

The pseudo-code representation of the algorithm described above are given in the annex 4 "Matrix-vector multiplication".

To calculate the transfer and computational complexities, we have to add the complexities of one parallel 2d decomposition, one parallel 2d best basis selection, one sequential 1d decomposition, one matrix vector multiplication, one sequential 1d recomposition and one array transfer.

If we consider the case when all the coefficients in the best basis are non negligible, this is of course much worse than the direct computation. But, if instead of all the coefficients N, we consider that only R are non negligible, the matrix-vector multiplication becomes of order O(R), and if R << N then this method is more efficient than the ordinary multiplication.

8.6. APPLICATIONS OF THE WAVELET PACKET TRANSFORM TO THE COMPRESSION AND ANALYSIS OF A TWO-DIMENSIONAL TURBULENT FLOW.

8.6.1. Methodology

We will now apply the wavelet packet transform algorithm to compress a two-dimensional turbulent field which has been computed using a pseudo-spectral Galerkin integration of the Navier-Stokes equations. We will represent this field as a sequence $S_{k,l}$ where $0 \leq k < 2^{Xmax}$ and $0 \leq l < 2^{Ymax}$.

Our first goal will be to decompose this turbulent field into two distinctive parts : its inhomogenous and organized components $S^{>}$, called coherent structures, and its homogenous and random components $S^{<}$ called background noise. Both components contain energy at all scales, so finding an optimal representation to perform this segmentation is not straightforward. An example is the Fourier representation; although the spatial information is kept in the phases, it is quite complicated to use it in that form. The grid-point representation is not a good one either, because we want to be able to compute the power spectrum of both components. The decay of energy at small scales, or the scaling law, provides an information very relevant for the stastistical theory of turbulence. To segment the field, we will look for the fewest number of wavelet packet coefficients necessary to adequately describe the coherent structures. We will select the fewest wavelet packet coefficients such that the discarded is homogeneous and its Fourier spectrum is a broad-band noise.

Our second goal will be to study an individual coherent structure. From the non compressed wavelet packet representation of the field, we will extract one coherent structure and compute its local spectrum to analyze its scaling law. The number of wavelet packet coefficients defining this structure will then give us an estimate of the number of degrees of freedom attached that it has. This is a kind of "theoretical dimension" of the coherent structure.

8.6.2. The compression technique

From the wavelet packet best-basis representation of the field, namely a family of kept wavelet packets ${}^{f_J}C^{j}$ with $j \in J$ (where $J \subset \mathrm{N}_{nmax} = \{n \in \mathrm{N}; 0 \leq n \leq nmax\}$) and $f_j \in F_J$ (where for each $j \in J$, $F_j \subset \mathrm{N}_{4^J-1}$), we will only keep the most energetic coefficients containing a given percentage of the total energy.

In sequential, the compression algorithm is implemented as follows : first, we sort the squares of all the wavelet packet coefficients in a decreasing order. Then, we keep as many coefficients as needed in order that the remaining field has a certain percentage of the initial energy (sum of the squares of the field's coefficients). This gives us a deterministic way of selecting a threshold ε underneath which we set all the wavelet packet coefficients to zero.

The parallel algorithm is not very different : in a first step, each process will sort the squares of the wavelet packet coefficients that belong to them. Then, two by two they will exchange their sorted list, and will perform a merge sort on those two lists to obtain the sorted list corresponding to both processes. They will repeat this step until each process has the whole sorted list of coefficients. This exchange of data is done as presented in paragraph 8.4 in the function *Local_array_transfer*, the difference is that here we merge the two arrays instead of adding them. In a last part, each process selects the coefficients as in the sequential algorithm and only keeps those that belong to itself.

The pseudo-code algorithm is given in the annex 5 "Compression".

8.6.3. The local extraction and local spectrum

The purpose of local extraction is to keep the wavelet packet coefficients ${}^{f}C^{j}_{k,l}$ that are positioned at a certain point (a_0, b_0) where $.(a_0, b_0) \in \left[0, 2^{Xmax}\right[\times \left[0, 2^{Ymax}\right[$, or in a certain interval $[a_1, a_2] \times [b_1, b_2]$ where $[a_1, a_2] \times [b_1, b_2] \subset \left[0, 2^{Xmax}\right[\times \left[0, 2^{Ymax}\right[$.

Each wavelet packet coefficient ${}^{f}C^{j}_{k,l}$ is localized in the space domain in the interval $\left[k_0 - 2^{j-1}, k_0 + 2^{j-1}\right] \times \left[l_0 - 2^{j-1}, l_0 + 2^{j-1}\right]$ where (k_0, l_0) is the center of mass of the wavelet packet coefficient and is defined as follows :

$$\begin{cases} k_0 = 2^j \cdot k + {}^{f}ShiftX^{j} \\ l_0 = 2^j \cdot l + {}^{f}ShiftY^{j} \end{cases} \quad \text{and}$$

$$\begin{cases} {}^{f}ShiftX^{j} = 2 * {}^{f/4}ShiftX^{j-1} + F(f \bmod 4, f/4)_{xshift} \\ {}^{f}ShiftY^{j} = 2 * {}^{f/4}ShiftY^{j-1} + F(f \bmod 4, f/4)_{yshift} \end{cases} \quad \text{and}$$

$$\begin{cases} F(d,f)_{xshift} = \dfrac{\sum\limits_{i,j} i * F(d,f)^2_{i,j}}{\sum\limits_{i,j} F(d,f)^2_{i,j}} \\ F(d,f)_{yshift} = \dfrac{\sum\limits_{i,j} j * F(d,f)^2_{i,j}}{\sum\limits_{i,j} F(d,f)^2_{i,j}} \end{cases}$$

We will keep this coefficient if the intersection between this interval and the interval $\left[a_0 - 2^{j-1}, a_0 + 2^{j-1}\right] \times \left[b_0 - 2^{j-1}, b_0 + 2^{j-1}\right]$ (respectively the interval $\left[a_1 - 2^{j-1}, a_2 + 2^{j-1}\right] \times \left[b_1 - 2^{j-1}, b_2 + 2^{j-1}\right]$) is not empty.

Extracting the local coefficients consists in keeping, for each wavelet packet belonging to the best-basis, the coefficients which verify this property.

To perform the local extraction, the algorithm will go over the wavelet packet tree in a depth-first order. This allows us to calculate ${}^{f}ShiftX^{\,j}$ and ${}^{f}ShiftY^{\,j}$ directly using the above formula. Then, for each kept wavelet packet of the tree, we will retain its local coefficients, namely those which are positionned in $(a_0\,,b_0)$ (respectively $[a_1\,,a_2]\times[b_1\,,b_2]$).

The pseudo-code algorithm is given in the annex 6 "Local extraction".

After having locally extracted some wavelet packet coefficients, we can compute their corresponding local spectrum. Once we have chosen what area of the field we want to represent in the Fourier domain, either at a point $(a_0\,,b_0)$ or in an interval $[a_1\,,a_2]\times[b_1\,,b_2]$, we perform the local extraction of that particular area and we reconstruct the filtered field by using only the coefficients kept during the extraction. Once the reconstruction is done, we compute the Fourier transform of this filtered field. We name it the local spectrum of the original field for the area $(a_0\,,b_0)$ (respectively $[a_1\,,a_2]\times[b_1\,,b_2]$) as it is the spectrum of the coefficients corresponding to that area.

8.6.4. Compression and analysis of a two-dimensional turbulent flow

The Navier-Stokes equations describe the dynamics of a two-dimensional incompressible flow.
Considering the periodic plane $S=(0,2\pi)\times(0,2\pi)\in \mathrm{R}^2$, and in the absence of external forcing, the Navier-Stokes equations are :

$$\begin{cases} \dfrac{\partial u}{\partial t}+(u\cdot\nabla)u+\nabla P-\nu\nabla^2 u=0 & in\, S\times R^+ \\ \nabla\cdot u=0 & in\, S\times R^+ \\ u(x,0)=u_0(x) & in\ S \end{cases}$$

where $\boldsymbol{u}$ is the velocity field, $\boldsymbol{P}$ the pressure field, and ν the kinematic viscosity.
We can rewrite these equations in terms of the vorticity $\omega=\dfrac{\partial u}{\partial x_1}-\dfrac{\partial u}{\partial x_2}$ and of the stream function ψ defined such that $u=\begin{pmatrix}u_1\\u_2\end{pmatrix}=\begin{pmatrix}-\dfrac{\partial\psi}{\partial x_2}\\ \dfrac{\partial\psi}{\partial x_1}\end{pmatrix}$.

The Navier-Stokes equations then become :

$$\left\{\begin{array}{ll}\dfrac{\partial\omega}{\partial t}+J(\psi,\omega)-\nu\nabla^2\omega=0 & in\, S\times R^+\\ \omega=\nabla^2\psi & in\, S\times R^+\\ \omega(x,0)=\omega_0(x) & in\, S\end{array}\right\} \text{ with } J(\psi,\omega)=\frac{\partial\psi}{\partial x_1}\cdot\frac{\partial\omega}{\partial x_2}-\frac{\partial\psi}{\partial x_2}\cdot\frac{\partial\omega}{\partial x_1}$$

The vorticity field we will consider was computed by Thierry Philipovitch[4] on a 512x512 grid, with a hyperdissipative operator $-(-\nabla^2)^4$ and no external forcing, using the numerical code developped by Claude Basdevant[4]. The time integration was done with an Adams-Bashforth scheme and the space integration with a pseudo-spectral Galerkin method. The code ran for 38000 time steps $\Delta t=10^{-3}\left(T^1\right)$, which corresponds to 635 initial eddy-turnover times, starting from a random distribution of vorticity with energy $E=0.5\left(M\,L^2\,T^{-2}\right)$ and an initial enstrophy $Z=279\left(T^2\right)$.

We first compressed the vorticity field and observed as shown in figure 8.6.4.2 that :

- the 3000 strongest wavelet packet coefficients, corresponding to a compression factor of 87, retain 99.9 % of the energy[5],
- the 739 strongest wavelet packet coefficients, corresponding to a compression factor of 355, retain 99 % of the energy,
- the 73 strongest wavelet packet coefficients, corresponding to a compression factor of 3591, retain 90 % of the energy,
- the 14 strongest wavelet packet coefficients, corresponding to a compression factor of 18724, retain 50 % of the energy.

[4] LMD, Ecole Normale Supérieure, Paris, France

[5] In turbulence the L^2-norm of the vorticity field is usually called enstrophy. In this paragraph, we have called it energy to conform with the notation of the first part of this chapter.

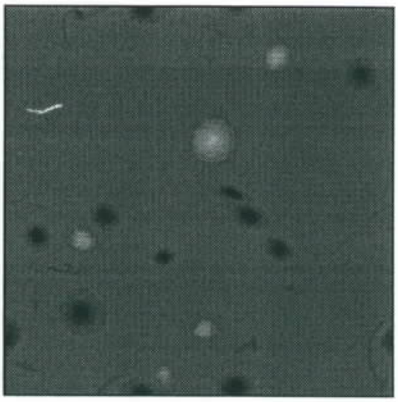
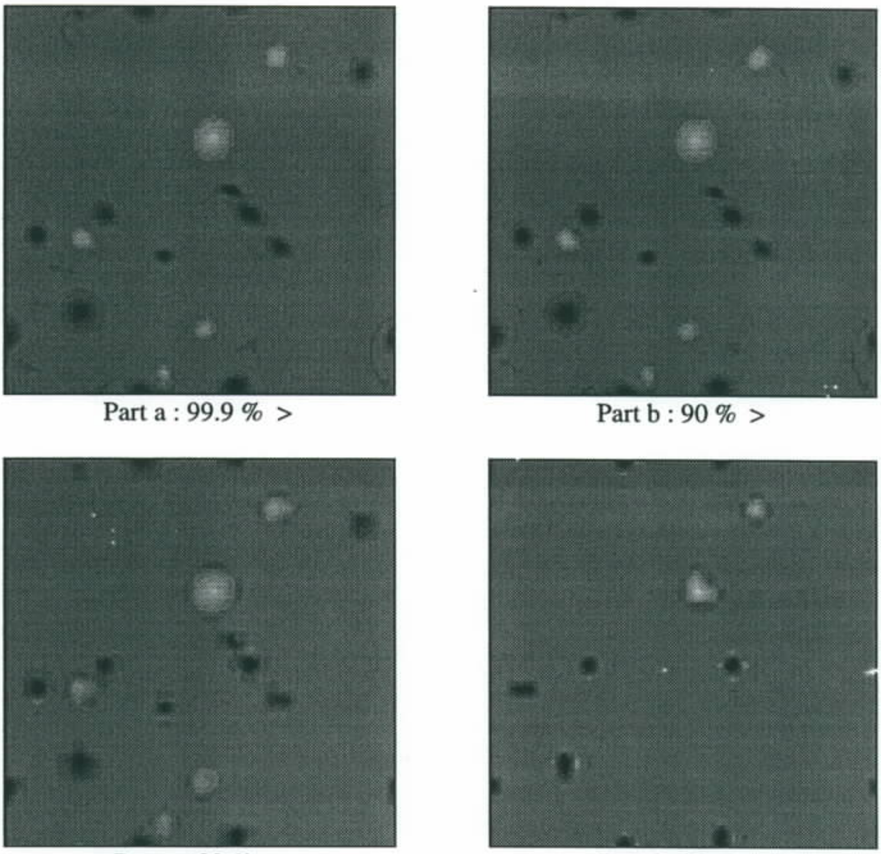


Figure 8.6.4.1 : Initial Vorticity field, 512x512 (262144) coefficients



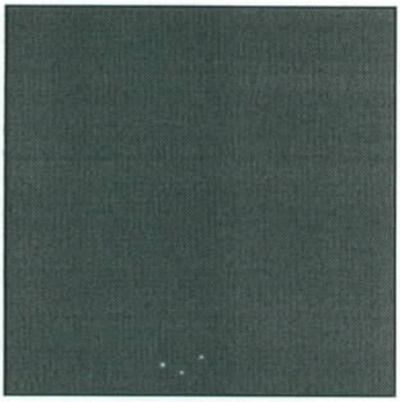
Part a : 99.9 % >

Part b : 90 % >

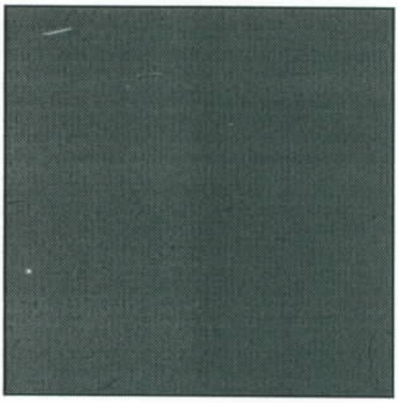
Part c : 99 % >

Part d : 50 % >

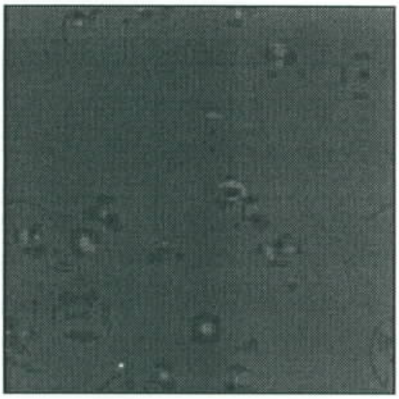
Figure 8.6.4.2 : The compressed fields $S^{>}$
Pallette : maximum 36.3 in white, minimum -38.1 in black



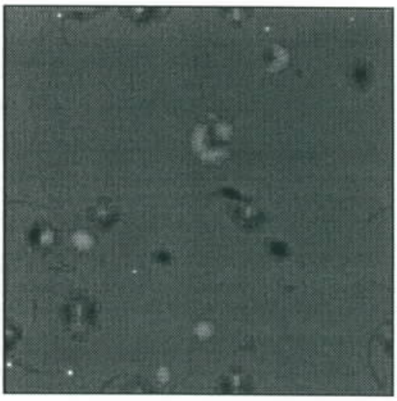
Part a : 0.1 % <



Part b : 1 % <



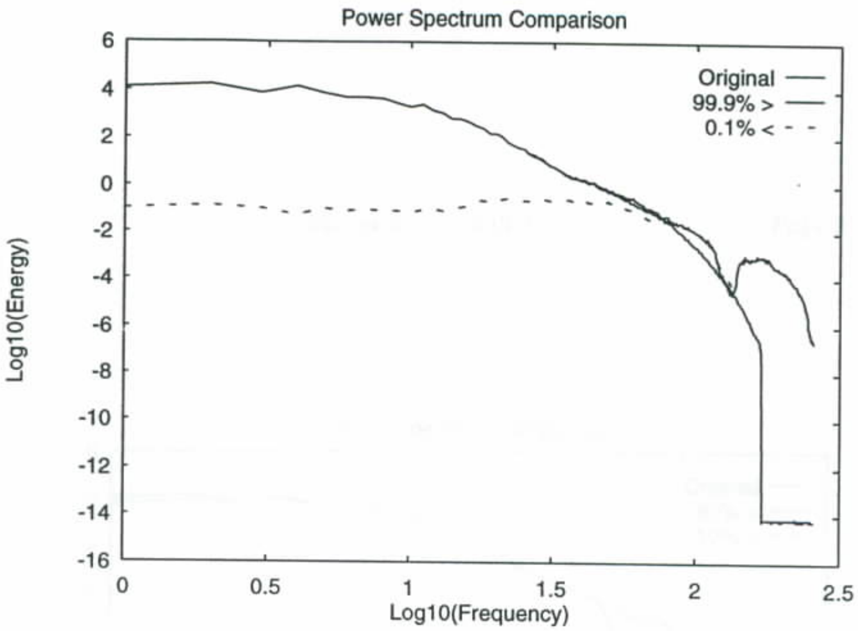
Part c : 10 % <

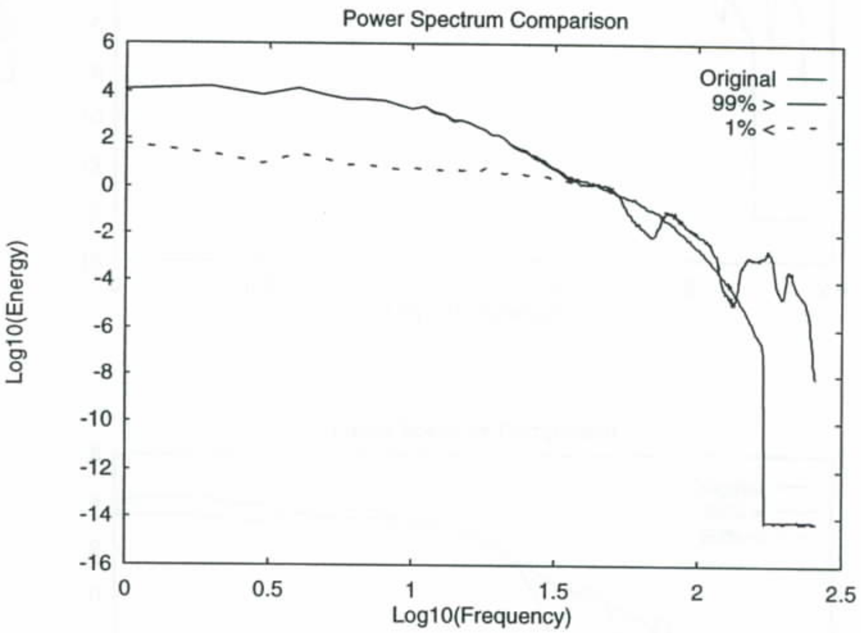


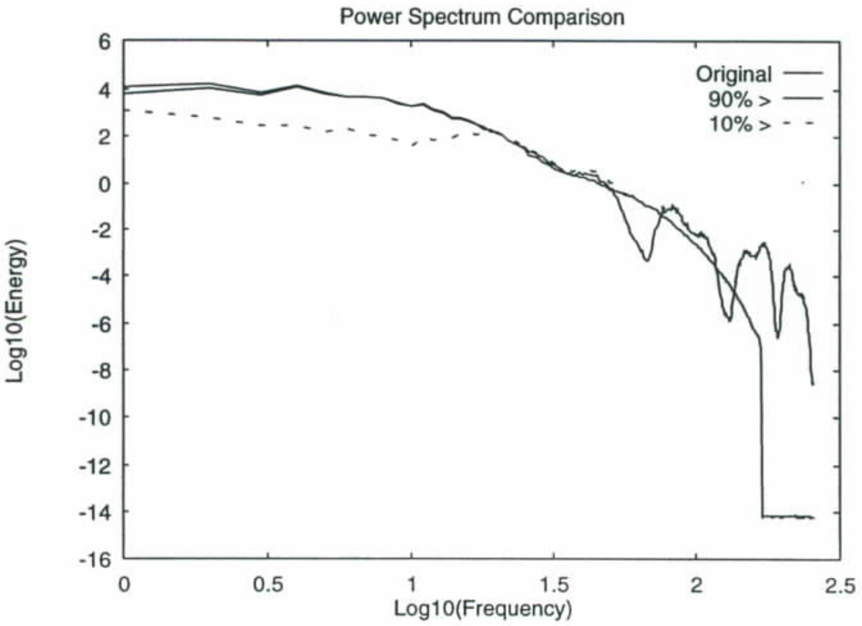
Part d : 50 % <

Figure 8.6.4.3 :
The discarded fields $S^<$
Pallette : maximum 36.3 in white, minimum -38.1 in black

In looking at the corresponding discarded fields and their power spectrum (Figure 8.6.4.3 and 8.6.4.4), the field containing 99% of the energy (compression factor of 355) is selected as the optimal one, because its discarded field remains quite a homogeneous broad-band noise. Therefore, the fewest number of wavelet packet coefficients necessary to represent the set of coherent structures of this vorticity field is 739, which gives us an estimate of the number of degrees of freedom we will have to consider to compute the flow evolution.


Power Spectrum Comparison
Log10(Energy)
Log10(Frequency)
Original
99.9% >
0.1% <
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
0
0.5
1
1.5
2
2.5


Power Spectrum Comparison
Log10(Energy)
Log10(Frequency)
Original
99% >
1% <
6
4
2
0
-2
-4
-6
-8
-10
-12
-14
-16
0
0.5
1
1.5
2
2.5



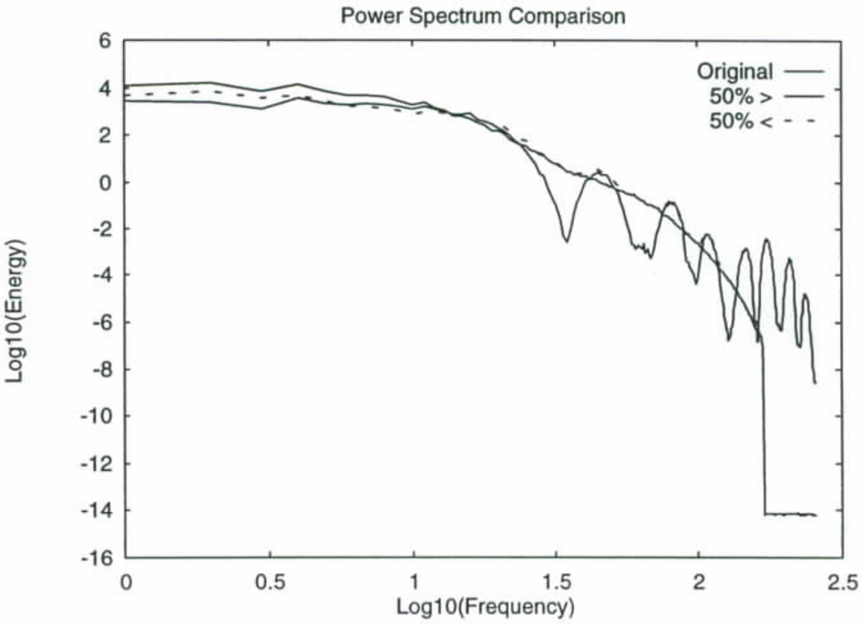


Figure 8.6.4.4 : $S^{>}$ and $S^{<}$ power spectrum comparisons

Second, We extracted the maximum positive coherent structure from the whole set of wavelet packet coefficients (Figure 8.6.4.5). There are 9000 wavelet packet coefficients defining the structure.

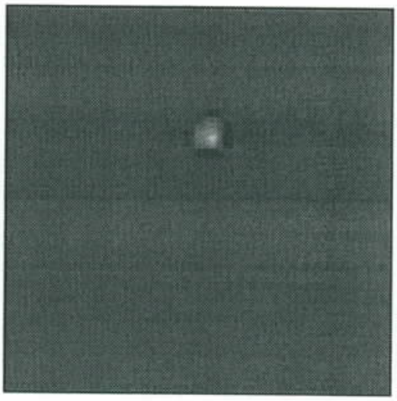


Figure 8.6.4.5 : Local extraction of the maximum positive vortex
Pallette : maximum 36.3 in white, minimum -38.1 in black

We then performed the fourier transform to this extracted field and we computed its power spectrum (Figure 8.6.4.6).

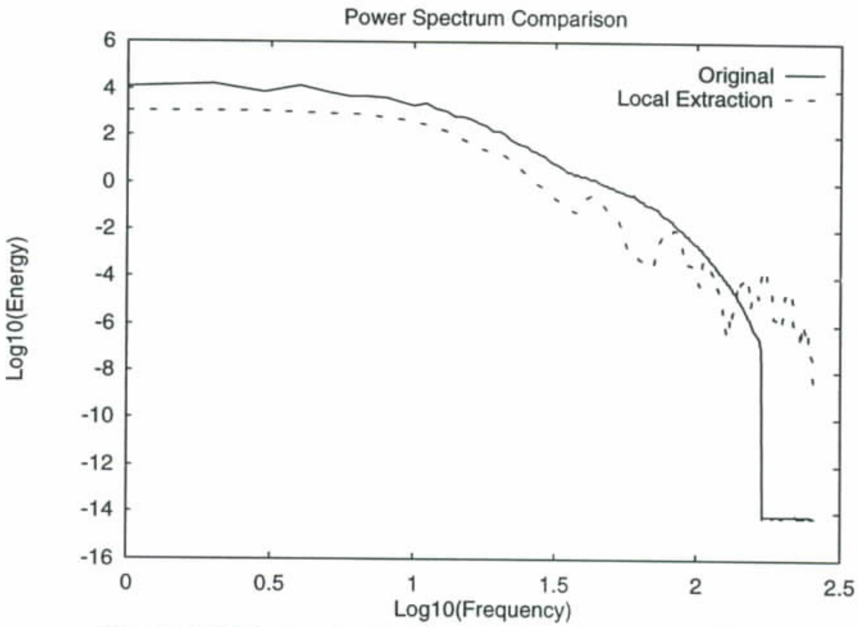


Figure 8.6.4.6 : Local extraction power spectrum comparison

8.7. ACKNOWLEDGEMENT

We are very grateful to Thierry Philipovitch for having provided us with the fully developed turbulent field we analyzed in this chapter.

8.8. REFERENCES

[1] Mladen Victor Wickerhauser, "Computation with Adapted Time-Frequency Atoms", pp. 175--184 in "Progress in Wavelet Analysis and Applications : Proceedings of the International Conference in Toulouse, 8--13 June 1992", edited by Yves Meyer and Sylvie Roques, Editions Frontières, ISBN 2-86332-130-7.

[2] Gregory Belkin, Ronald R. Coifman and Vladimir Rokhlin, "Fast Wavelet Transform and Numerical Algorithms I, "Communications on Pure and Applied Math, XLIV (1991), 141--183.

[3] Stephane G. Mallat, "Multiresolution Approximation and Wavelet Orthonormal Bases of $L^2(R)$", Transactions of the American Mathematical society, 315 (1989), 69--87.

[4] Ronald R. Coifman, Yves Meyer, Stephen R. Quake and Mladen Victor Wickerhauser, "Signal Processing and Compression with Wavelet Packets", pp. 77--93 in "Progress in Wavelet Analysis and Applications : Proceedings of the International Conference in Toulouse, 8--13 June 1992", edited by Yves Meyer and Sylvie Roques, Editions Frontières, ISBN 2-86332-130-7.

[5] Ronald R. Coifman and Mladen Victor Wickerhauser, "Entropy Based Algorithms for Best Basis Selection", IEEE Transactions on Information Theory, 32 (March 1992), 712--718.

8.9. ANNEXES : Algorithms written in pseudo-code

8.9.1. Annex 1 : Decomposition

The function doing the data transfers has the following representation in pseudo-code :

The parameters of function *Data transfers* are :

```
p is the process_name
jp is such that $4^{jp}$ is the number of processes
j is the current scale
f is the current frequency
width is the width of the signal at scale 0
height is the height of the signal at scale 0
Local_packet is the local packet to transfer
```

First we initialize the parameters :

```
f = p / $4^{jp-j}$
factor = 1
f_child = p
llx = width / $2^{jp}$
lly = height / $2^{jp}$
base_x = cx(p,jp,width)-cx(f,j,width)
base_y = cy(p,jp,height)-cy(f,j,height)

${}^{f}C^{j}_{base\_x+0..wh-1,base\_y+0..ht-1} = Local\_packet_{0..wh-1,0..ht-1}$
```

Then we transfer data :

```
FOR jj = jp-1 TO j BY STEP -1
  f_father = f_child / 4
  depl = 4 * f_father
  IF (((p/factor) equals depl) OR ((p/factor) equals (depl+2)))
      bx=cx(f_child*factor,jp,width)-cx(f,j,width)
      by=cy(f_child*factor,jp,height)-cy(f,j,height)
      Send ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ to process *p+factor*

      bx=cx((f_child+1)*factor,jp,width)-cx(f,j,width)
      by=cy((f_child+1)*factor,jp,height)-cy(f,j,height)
      Receive ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ from process *p+factor*
  ELSE
      bx=cx((f_child-1)*factor,jp,width)-cx(f,j,width)
```

```
            by=cy((f_child-1)*factor,jp,height)-cy(f,j,height)
            Receive ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ from process p-factor

            bx=cx(f_child*factor,jp,width)-cx(f,j,width)
            by=cy(f_child*factor,jp,height)-cy(f,j,height)
            Send ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ to process p-factor
        ENDIF

        llx = llx *2

        IF (((p/factor) equals depl) OR ((p/factor) equals (depl+1)))
            bx=cx(depl*factor,jp,width)-cx(f,j,width)
            by=cy(depl*factor,jp,height)-cy(f,j,height)
            Send ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ to process p+2*factor

            bx=cx((depl+2)*factor,jp,width)-cx(f,j,width)
            by=cy((depl+2)*factor,jp,height)-cy(f,j,height)
            Receive ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ from process p+2*factor
        ELSE
            bx=cx((depl-2)*factor,jp,width)-cx(f,j,width)
            by=cy((depl-2)*factor,jp,height)-cy(f,j,height)
            Receive ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ from process p-2*factor

            bx=cx(depl*factor,jp,width)-cx(f,j,width)
            by=cy(depl*factor,jp,height)-cy(f,j,height)
            Send ${}^{f}C^{j}_{bx+0..llx-1,by+0..lly-1}$ to process p-2*factor
        ENDIF

        lly = lly*2
        f_child = f_father
        factor = 4*factor
       ENDFOR j
```

Data transfers returns the packet ${}^{f}C^{j}$.

The algorithm of the decomposition when j < jp has the following pseudo-code implemetation :

```
       p is the process name
       $4^{jp}$ is the number of processes
       wh = width / $2^{jp}$
       ht = height / $2^{jp}$
```

```
FOR j = 0 TO jp-1
     f = p / $4^{jp-j}$
     fc = p / $4^{jp-j-1}$
     d = fc modulo 4

     //First we exchange data to have ${}^{f}C^{j}$
          ${}^{f}C^{j} = Data\ transfers\left(p, jp, j, f, 2^{Xmax}, 2^{Ymax}, {}^{f}IC^{j}\right)$

     //Second, we define the filter to use :
          FF = F(d,f)

     //Third, we calculate the x- and y- offsets :
          base_x = cx(p,jp,width)-cx(f,j,width)
          base_y = cy(p,jp,height)-cy(f,j,height)

     // Fourth we calculate ${}^{fc}IC^{j+1}$
          ${}^{fc}IC^{j+1}_{k,l} = \sum_{kk,ll} FF_{kk,ll}\ {}^{f}C^{j}_{kk+2*(base\_x+k),\, ll+2*(base\_y+l)}$
ENDFOR j
```

The algorithm of the decomposition when j ≥ jp has the following pseudo-code implemetation :

```
p is the process name
FOR j = jp TO nmax
     FOR f = $4^{j-jp}$ * p TO $4^{j-jp}$ * (p+1) - 1
          FOR d = 0 TO 3
               First we set the filter to use : FF = F(d,f)
               The we calcu!ate ${}^{4f+d}C^{j+1}$ :
                    ${}^{4f+d}C^{j+1}_{k,l} = \sum_{kk,ll} FF_{kk,ll}\ {}^{f}C^{j}_{kk+2*k,\, ll+2*l}$
          ENDFOR d
     ENDFOR f
ENDFOR j
```

8.9.2. Annex 2 : Best Basis Selection

First, we consider the case when j ≥ jp :

```
FOR f = $4^{nmax-jp}$ * p TO $4^{nmax-jp}$ * (p+1) - 1
     ${}^{f}C^{j}_{status} = KEPT$
     ${}^{f}C^{j}_{value} = M\left({}^{f}C^{j}\right)$
```

```
ENDFOR f

FOR j = nmax-1 TO jp BY STEP -1
    FOR f = $4^{j-jp}$ * p TO $4^{j-jp}$ * (p+1) - 1
        $v_1 = \sum_{d=0}^{3} {}^{4*f+d}C_{value}^{j+1}$
        $v_0 = M\left({}^{f}C^{j}\right)$
        IF ($v_0 \le v_1$)
            ${}^{f}C_{status}^{j} = KEPT$
            ${}^{f}C_{value}^{j} = v_0$
            factor=1
            FOR jj=j+1 TO nmax
                factor = factor*4
                f_base = factor*f
                FOR ff = 0 TO factor
                    ${}^{f\_base+ff}C_{status}^{jj} = NOT\_KEPT$
                ENDFOR ff
            ENDFOR jj
        ELSE
            ${}^{f}C_{status}^{j} = NOT\_KEPT$
            ${}^{f}C_{value}^{j} = v_1$
        ENDIF
    ENDFOR f
ENDFOR j
```

Second, we consider the case when j < jp :
To implement this algorithm,

First set the parameters of function *Local value transfer* :

p is the name of the current process
np is the number of processes doing the transfer
p_base is the pivot process
p_shift is the shift of processes to do the transfer
local_value_p is the value of the process p to transfer

We must also store some side data :

```
test=np
stepsize=p_shift
pvalue=$\text{local\_value}_p$
```

Then, we do the transfers :

```
WHILE (test > p_shift)
    IF (((p - p_base) mod (2*stepsize)) < stepsize)
```

```
            p2=p+stepsize
            start=1
        ELSE
            p2=p-stepsize
            start=0
        ENDIF

        IF (start = 1)
            Send pvalue to process p2
            Receive p2value from process p2
        ELSE
            Receive p2value from process p
            Send pvalue to process p
        ENDIF

        pvalue = pvalue + p2value

        test = test/2
        stepsize = stepsize * 2
    ENDWHILE
    total_value=pvalue
```

Local value transfer returns total_value

The best basis selection algorithm when j is smaller than jp is then :

```
p is the name of the current process
$4^{jp}$ is the maximum number of processes

FOR j = jp-1 TO 0 BY STEP -1
    f = p / $4^{jp-j}$
    f_child = p / $4^{jp-j-1}$

    $v_0$ = Local value transfer(p,$4^{jp-j}$,f,1,$M\left({}^{f}lC^{j}\right)$)
    $v_1$ = Local value transfer(p,$4^{jp-j}$,f,$4^{jp-j-1}$,${}^{f\_child}lC^{j+1}_{value}$ )

    IF ($v_0 \leq v_1$)
        ${}^{f}lC^{j}_{status} = KEPT$
        ${}^{f}lC^{j}_{value} = v_0$
        factor=1
        FOR jj=j+1 TO nmax
            factor = factor*4
            f_base = factor*f
            FOR ff = 0 TO factor
                ${}^{f\_base+ff}lC^{jj}_{status} = NOT\_KEPT$
            ENDFOR ff
        ENDFOR jj
    ELSE
        ${}^{f}lC^{j}_{status} = NOT\_KEPT$
        ${}^{f}lC^{j}_{value} = v_1$
    ENDIF
ENDFOR j
```

8.9.3. Annex 3 : Reconstruction

First, we consider the case when j > jp.
In this case, the algorithm is the same as the sequential one :

```
p is the process name
FOR j = nmax TO jp+1 BY STEP -1
    FOR f = $4^{j-jp-1}$ * p TO $4^{j-jp-1}$ * (p+1) - 1
        IF (${}^{f}C^{j-1}$ exists)
            IF (${}^{f}C^{j-1}_{status} = NOT\_KEPT$)
                We set ${}^{f}C^{j-1}_{k,l} = 0, \forall k,l$
            ENDIF
        ENDIF
    ENDFOR f

    FOR f = $4^{j-jp}$ * p TO $4^{j-jp}$ * (p+1) -1
        IF (${}^{f}C^{j}$ exists)
            IF (${}^{f}C^{j}_{status} = KEPT$)
                1/ We set the filter to use for reconstructing :
                    FF = F(f mod 4,f / 4)
                2/ We create ${}^{f/4}C^{j-1}$ is it doesn't exist
                    by setting all its coefficients to zero.
                3/ We set ${}^{f/4}C^{j-1}_{status} = KEPT$
                4/ We compute the partial reconstruction :
                $ {}^{f/4}C^{j-1}_{k,l} = {}^{f/4}C^{j-1}_{k,l} + \sum_{n,m} FF_{k-2n,l-2m} * {}^{f}C^{j}_{n,m} $
                5/ We don't need any more ${}^{f}C^{j}$
            ENDIF
        ENDIF
        ENDFOR f
    ENDFOR j
```

Second, we consider the case when scale $j \leq jp$.

```
    Function *Local array transfer* has the following parameters :
        p is the name of the current process
        np is the number of processes doing the transfer
        p_base is the pivot process
        p_shift is the shift of processes to do the transfer
        local_array$_p$ is the array of the process p to transfer
        size_array is the size of the local array

    It contains the following local variables :
        test=np
        stepsize=p_shift
        pvalue[i]=local_value$_p$[i] for $0 \leq i < size\_array$

    It then does these transfers :
        WHILE (test > p_shift)
            IF (((p - p_base) mod (2*stepsize)) < stepsize)
                p2=p+stepsize
                start=1
            ELSE
                p2=p-stepsize
                start=0
            ENDIF

            IF (start = 1)
                Send array pvalue to process p2
                Receive array p2value from process p2
            ELSE
                Receive array p2value from process p
                Send array pvalue to process p
            ENDIF

            pvalue[i] = pvalue[i] + p2value[i] for $0 \leq i < size\_array$

            test = test/2
            stepsize = stepsize * 2
        ENDWHILE

        total_value[i] =pvalue[i]
    *Local array transfer* returns the array total_value
```

The reconstruction algorithm can be implemented as follows :

```
p is the name of the process
$2^{Xmax}*2^{Ymax}$ is the size of the field at scale 0
$wh = 2^{Xmax-jp}$
$ht = 2^{Ymax-jp}$

FOR j = jp TO 1 BY STEP -1
    $np = 4^{jp-j}$
    $f = p / 4^{jp-j}$
    $f2 = f / 4$
    IF ($^f lC^j$ exists)
        IF ($^f lC^j_{status} = NOT\_KEPT$)
            1/ We set $^f lC^j_{k,l} = 0,\ \forall 0 \le k < wh,\ 0 \le l < ht$
            2/ We set $^f lC^j_{status} = KEPT$
        ENDIF
    ELSE
        1/ We create $^f lC^j$ by setting all its coefficients to zero
        2/ We set $^f lC^j_{status} = KEPT$
    ENDIF

    IF ($^f lC^j$ exists)
        IF ($^f lC^j_{status} = KEPT$)
            1/ We calculate local shifts
            $bx = cx(p, jp, 2^{Xmax}) - cx(f2, j-1, 2^{Xmax})$
            $by = cy(p, jp, 2^{Ymax}) - cy(f2, j-1, 2^{Ymax})$
            $base\_x = cx(p, jp, 2^{Xmax}) - cx(f, j, 2^{Xmax})$
            $base\_y = cy(p, jp, 2^{Ymax}) - cy(f, j, 2^{Ymax})$

            2/ We create a real size temporary father $^{f2} tC^{j-1}$
            with null coefficients

            3/ We update it if needed
            IF ($^{f2} lC^{j-1}$ exists)
                IF ($^{f2} lC^{j-1}_{status} = KEPT$)
                $^{f2} tC^{j-1}_{bx+0..wh-1,by+0..ht-1} = {}^{f2} lC^{j-1}_{0..wh-1,0..ht-1}$
                ENDIF
            ENDIF
            4/ We calculate the partial reconstruction
        $^{f2}tC^{j-1}_{k,l} = {}^{f2}tC^{j-1}_{k,l} + \sum_{n,m} FF_{k-2*(n+base\_x),\, l-2*(m+base\_y)} * {}^{f}lC^{j}_{n,m}$

            5/ We calculate the sum of all $^{f2}tC^{j-1}$
            len $= 2^{Xmax-j+1} * 2^{Xmax-j+1}$
    $^{f2}tC^{j-1} = Local\ arraytransfer\left(p, 4^{jp-j+1}, f2*4^{jp-j+1}, 1, {}^{f2}tC^{j-1}, len\right)$

            6/ If $^{f2}lC^{j-1}$ doesn't exist we set it to the null vector
            7/ We set $^{f2}lC^{j-1}_{status} = KEPT$
            8/ We transfer the data from $^{f2}tC^{j-1}$ to $^{f2}lC^{j-1}$
                $^{f2}lC^{j-1}_{0..wh-1,0..ht-1} = {}^{f2}tC^{j-1}_{bx+0..wh-1,by+0..ht-1}$

            9/ We don't need any more $^{f2}tC^{j-1}$ and $^{f}lC^{j}$
        ENDIF
    ENDIF
ENDFOR
```

8.9.4. Annex 4 : Matrix-Vector Multiplication

The multiplication algorithm may be implemented as follows :

```
p is the process name
$2^{Xmax}$ by $2^{Ymax}$ is the size of the matrix C
$2^{Xmax}$ is the size of the vector D
so $2^{Ymax}$ is the size of the vector E

wh=$2^{Xmax-jp}$ and ht=$2^{Ymax-jp}$

A/ In parallel, using all the processes
    We decompose C in all the 2D-wavelet packets,
    We select the best basis for C

B/ On each process,
    We decompose D in all the 1D-wavelet packets

C/ We do the multiplication when a packet is
    shared by more than one process
FOR j = 0 TO jp-1
    f = p / $4^{jp-j}$
```

```
        IF ($^{f}lC^{j}$ exists) THEN
            IF ($^{f}lC^{j}_{status} = KEPT$) THEN
                1/ We set the signal shifts
                $base\_x = cx\left(p, jp, 2^{Xmax}\right) - cx(f, j, 2^{Xmax})$
                $base\_y = cy\left(p, jp, 2^{Ymax}\right) - cy(f, j, 2^{Ymax})$

                2/ We set the frequencies among the x- and y- axis
                $fx = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left( \left[ \frac{f}{4^{j-jj-1}} \right] \bmod 2 \right)$
                $fy = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left( \left[ \frac{\left[ \frac{f}{4^{j-jj-1}} \right]}{2} \right] \bmod 2 \right)$

                3/ We keep the local vector
                $^{fx}lD^{j}_{0..wh-1} = {}^{fx}D^{j}_{base\_x+0..wh-1}$

                4/ We perform the smaller matrix-vector multiplication
                $^{fy}lE^{j} = {}^{f=(fx,fy)}lC^{j} \times {}^{fx}lD^{j}$

                5/ We insert $^{fy}lE^{j}$ into $^{fy}E^{j}$
                $^{fy}E^{j}_{base\_y+0..ht-1} = {}^{fy}lE^{j}_{0..ht-1}$

                6/ We set $^{fy}E^{j}_{status} = KEPT$
            ENDIF
        ENDIF
    ENDFOR j
```

D/ We do the multiplication when a packet belongs to a process

```
FOR j = jp TO nmax
    FOR f = $4^{j-jp}$ * p TO $4^{j-jp}$ * (p+1) -1
        IF ($^{f}C^{j}$ exists) THEN
            IF ($^{f}C^{j}_{status} = KEPT$) THEN
                1/ We perform the smaller
                matrix-vector multiplication
                $^{fy}E^{j} = {}^{f=(fx,fy)}C^{j} \times {}^{fx}D^{j}$
```

```
                    2/ We set ${}^{fy}E^{j}_{status} = KEPT$
                ENDIF
            ENDIF
        ENDFOR f
ENDFOR j
```

E/ On each process,
    We Reconstruct from all the kept wavelet packets ${}^{fy}E^{j}$
    Each process has its own part of the partial vector E

F/ We exchange and combine the partial vectors E to have the whole vector E
    $E = Local\,array\,transfer\left(p, 4^{jp}, 0, 1, E, 2^{Ymax}\right)$

G/ The multiplication is finished,
    Each process has the whole vector E.

8.9.5. Annex 5 : Compression

Here is first the function doing the merge of the local sorted list of wavelet packet coefficients :

Function *Merge local list* has the following parameters :
- p is the name of the current process
- np is the number of processes doing the transfer
- p_base is the pivot process
- p_shift is the shift of processes to do the transfer
- local_list$_p$ is the list of the process p to transfer
- size_list is the size of the local list

It contains the following local variables :
- test=np
- stepsize=p_shift
- plist[i] = local_list$_p$[i] for $0 \le i < size\_list$

It then does these transfers :

```
WHILE (test > p_shift)
    IF (((p - p_base) mod (2*stepsize)) < stepsize)
        p2=p+stepsize
        start=1
    ELSE
        p2=p-stepsize
```

```
            start=0
        ENDIF

        IF (start = 1)
            Send size_list to process p2
            Send list plist to process p2
            Receive size_list2 to process p2
            Receive list  p2list from process p2
        ELSE
            Receive size_list2 to process p2
            Receive list  p2list from process p2
            Send size_list to process p2
            Send list plist to process p2
        ENDIF

        // We perform a merge sort in order that pvalue be the sorted
         list corresponding to plist and p2list
        plist = merge(plist, p2list)
        size_list = size_list + size_list2

        test = test/2
        stepsize = stepsize * 2
    ENDWHILE

    total_list[i] =plist[i] for $0 \le i < size\_list$
```

Merge local list returns total_list.

Then here is the pseudo-code for the compression algorithm :

p is the process name
4^{jp} is the number of processes
$2^{Xmax} \times 2^{Ymax}$ is the size of the field at scale 0
max_energy if the sum of the square of the coefficients that we would like to keep. (For example, a certain percentage of the initial field energy).

1/ First, the process groups all its wavelet packet coefficients and sort them such as their squares be in a decreasing order. The process has now a list $L1 = \left({}^{f_0}C^{j_0}_{k_0,l_0}, {}^{f_1}C^{j_1}_{k_1,l_1}, \ldots, {}^{f_{nl-1}}C^{j_{nl-1}}_{k_{nl-1},l_{nl-1}} \right)$ such that all the coefficients verify $\left({}^{f_0}C^{j_0}_{k_0,l_0} \right)^2 > \left({}^{f_1}C^{j_1}_{k_1,l_1} \right)^2 > \ldots > \left({}^{f_{nl-1}}C^{j_{nl-1}}_{k_{nl-1},l_{nl-1}} \right)^2$.

```
2/ It exchanges its list with other processes until it has the entire sorted list.
    $TL = Merge\ local\ list\left(p, 4^{jp}, 0, 1, L1, n1\right)$
    $TL = \left( {}^{f_0}TC^{j_0}_{k_0, l_0}, {}^{f_1}TC^{j_1}_{k_1, l_1}, \dots, {}^{f_{tn-1}}TC^{j_{tn-1}}_{k_{tn-1}, l_{tn-1}} \right)$

3/ It selects only the coefficients needed to have the specified max_energy.
    sum = 0
    nc = 0
    WHILE ( (sum < max_energy) AND (nc < tn) )
        $sum = sum + \left( {}^{f_{nc}}C^{j_{nc}}_{k_{nc}, l_{nc}} \right)^2$
        nc = nc + 1
    ENDWHILE

4/ It keeps only the coefficients that belong to itself
    FOR i = 0 TO nc-1
        IF (${}^{f_i}C^{j_i}$ exists)
            IF (${}^{f_i}C^{j_i}_{status} = KEPT$)
                1/ We create ${}^{f_i}CC^{j_i}$ to the null vector if it does
                not exist

                2/ We set the coefficient to keep
                    ${}^{f_i}CC^{j_i}_{k_i, l_i} = {}^{f_i}TC^{j_i}_{k_i, l_i}$
            ENDIF
        ENDIF
    ENDFOR

5/ The new set of wavelet packets ${}^{f}CC^{j}$ represent the compressed version of the initial field ${}^{f}C^{j}$.
```

8.9.6. Annex 6 : Local extraction

First, here is the pseudo-code for the function doing the sequential extraction on the interval $[a_1, a_2] \times [b_1, b_2]$ (We consider the extraction at a point by setting a_2 to a_1 and b_2 to b_1) :

```
Function Extract_local_Coefficients has the following parameters :
    j is the scale and
    f is the frequency defining the wavelet packet to consider
    ShiftX and ShiftY are the x- and y- shifts at scale j-1 and frequency
```

f/4.

First, we compute the x- and y- shifts for the considered packet

```
    $ShiftX = 2*ShiftX + F(f \bmod 4, f/4)_{xshift}$
    $ShiftY = 2*ShiftY + F(f \bmod 4, f/4)_{yshift}$
```

Then, if the wavelet packet is part of the best basis, we extract its local coefficients

```
    IF (${}^{f}C^{j}$ exists)
        IF (${}^{f}C^{j}_{status} = KEPT$)
            1/ We initialize ${}^{f}CL^{j}$ to the null vector
            2/ We set its status to KEPT
                ${}^{f}CL^{j}_{status} = KEPT$
            3/ We set the boundaries for the local extraction
            // Round(x) returns the nearest integer to x
            $ipx1 = Round\left(\frac{a_1 - 2^{j-1} - ShiftX}{2^j}\right)$
            $ipx2 = Round\left(\frac{a_2 + 2^{j-1} - ShiftX}{2^j}\right)$
            $ipy1 = Round\left(\frac{b_1 - 2^{j-1} - ShiftY}{2^j}\right)$
            $ipy2 = Round\left(\frac{b_2 + 2^{j-1} - ShiftY}{2^j}\right)$
            4/ We keep only the local coefficients
            FOR ipx = ipx1 TO ipx2
                FOR ipy = ipy1 TO ipy2
        ${}^{f}CL^{j}_{ipx \bmod 2^{Xmax-j},\, ipy \bmod 2^{Ymax-j}} = {}^{f}C^{j}_{ipx \bmod 2^{Xmax-j},\, ipy \bmod 2^{Ymax-j}}$
                ENDFOR
            ENDFOR
        ENDIF
    ENDIF
```

In a last part, we recursively extract the local coefficients in the child packets

```
    IF ($j \neq nmax$)
        FOR d = 0 TO 3
        $Extract\_Local\_Coefficients(j+1, 4*f+d, ShiftX, ShiftY)$
        ENDFOR
    ENDIF
```

The result is the new set of wavelet packets ${}^{f}CL^{j}$

Then, here is the pseudo-code algorithm for extracting the interval $[a_1, a_2] \times [b_1, b_2]$ ($a_1 \le a_2$ and $b_1 \le b_2$) on each processor (We consider the extraction at a point by setting a_2 to a_1 and b_2 to b_1) :

```
p is the process name
$4^{jp}$ is the number of processes
$2^{Xmax} \times 2^{Ymax}$ is the size of the field at scale 0

First, we consider the case when j < jp
    IF (${}^{0}lC^{0}$ exists)
        IF (${}^{0}lC^{0}_{status} = KEPT$)
            1/ We initialize ${}^{0}lCL^{0}$ to the null vector
            2/ ${}^{0}lCL^{0}_{status} = KEPT$
            3/ We compute the base coordinates
                $bx = cx\left(p, jp, 2^{Xmax}\right)$
                $by = cy\left(p, jp, 2^{Ymax}\right)$
            4/ We set the boundaries of the local extraction
                $OK = 1$
                $ipx1 = \max(a_1 - bx, 0)$
                IF ($ipx1 \ge 2^{Xmax-jp}$) $OK = 0$
                $ipy1 = \max(b_1 - by, 0)$
                IF ($ipy1 \ge 2^{Ymax-jp}$) $OK = 0$
                $ipx2 = \min(a_2 - bx, 2^{Xmax-jp} - 1)$
                IF ($ipx2 < 0$) $OK = 0$
                $ipy2 = \min(b_2 - by, 2^{Ymax-jp} - 1)$
                IF ($ipy2 < 0$) $OK = 0$
            5/ We extract the local coefficients
                IF ($OK = 1$)
                    FOR ipx = ipx1 TO ipx2
                        FOR ipy = ipy1 TO ipy2
                            ${}^{f}lCL^{j}_{ipx,ipy} = {}^{f}lC^{j}_{ipx,ipy}$
                        ENDFOR
                    ENDFOR
                ENDIF
        ENDIF
    ENDIF
```

```
$ShiftX = 0$
$ShiftY = 0$
IF ($^0lCL^0$ does not exist)
    FOR j = 1 TO jp-1
        f = p / 4^jp
        $ShiftX = 2 * ShiftX + F(f \bmod 4, f/4)_{xshift}$
        $ShiftY = 2 * ShiftY + F(f \bmod 4, f/4)_{yshift}$
        IF ($^f lC^j$ exists)
            IF ($^f lC^j_{status} = KEPT$)
            1/ We initialize $^f lCL^j$ to the null vector
            2/ $^f lCL^j_{status} = KEPT$
            3/ We compute the base coordinates
                $bx = cx(p, jp, 2^{Xmax}) - cx(f, j, 2^{Xmax})$
                $by = cy(p, jp, 2^{Ymax}) - cy(f, j, 2^{Ymax})$
            4/ We set the boundaries of the local extraction
                $OK = 1$
                $ipx1 = Round\left(\frac{a_1 - 2^{j-1} - ShiftX}{2^j}\right)$
                $ipx1 = \max\left(\left(ipx1 \bmod 2^{Xmax-j}\right) - bx, 0\right)$
                IF ($ipx1 \geq 2^{Xmax-jp}$) $OK = 0$
                $ipy1 = Round\left(\frac{b_1 - 2^{j-1} - ShiftY}{2^j}\right)$
                $ipy1 = \max\left(\left(ipy1 \bmod 2^{Ymax-j}\right) - by, 0\right)$
                IF ($ipy1 \geq 2^{Ymax-jp}$) $OK = 0$
                $ipx2 = Round\left(\frac{a_2 + 2^{j-1} - ShiftX}{2^j}\right)$
                $ipx2 = \min\left(\left(ipx2 \bmod 2^{Xmax-j}\right) - bx, 2^{Xmax-jp}\right)$
                IF ($ipx2 < 0$) $OK = 0$
                $ipy2 = Round\left(\frac{b_2 + 2^{j-1} - ShiftY}{2^j}\right)$
                $ipy2 = \min\left(\left(ipy2 \bmod 2^{Ymax-j}\right) - by, 2^{Ymax-jp}\right)$
                IF ($ipy2 < 0$) $OK = 0$
            5/ We extract the local coefficients
```

```
                IF ($OK=1$)
                    FOR ipx = ipx1 TO ipx2
                        FOR ipy = ipy1 TO ipy2
                            ${}^{f}lCL^{j}_{ipx,ipy} = {}^{f}lC^{j}_{ipx,ipy}$
                        ENDFOR
                    ENDFOR
                ENDIF
            ENDIF
        ENDIF
    ENDFOR
ENDIF
```

Then, the case when j ≥ jp

$Extract\_Local\_Coefficients(jp, p, ShiftX, ShiftY)$

The result is the new set of wavelet packets ${}^{f}CL^{j}$ and ${}^{f}lCL^{j}$