CHAPTER 8 : A PARALLEL TWO DIMENSIONAL WAVELET PACKET TRANSFORM AND ITS APPLICATION TO MATRIX-VECTOR MULTIPLICATION.

Eric GOIRAND
Mladen Victor WICKERHAUSER

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

Abstract

In a first part, we study a parallel algorithm (on 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.

8.1. INTRODUCTION

That's the introduction.

8.2. A PARALLEL WAVELET PACKET DECOMPOSITION

8.2.1. Definition

From a two-dimensionnal periodic signal $S = (s_{i,j})$ with $0 \leq i < 2^{Xmax}$ and $0 \leq j < 2^{Ymax}$, we pose nmax = min(Xmax,Ymax). In a first step we want to

calculate all the wavelet packets ${}^{f_j}C^j$, 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 all other packet is defined recursively by the formula :

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

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 : Filter(0) = HH, Filter(1) = HG, Filter(2) = GH and Filter(3) = GG. In order to keep the frequencies (among the x and y axis) increasing, we then define $F(d,f_j)_{n,m} = Filter(gray\_code2d(d,f_j))_{n,m}$ where $gray\_code2d(d,f_j) = 2\cdot a + b$ wit

$$a = \begin{cases} d \bmod 2 if\ f_j is even \\ 1-(d \bmod 2) if\ f_j is odd \end{cases} \text{ and } b = \begin{cases} \left[\frac{d}{2}\right] if \left[\frac{f_j}{2}\right] is even \\ 1-\left[\frac{d}{2}\right] if \left[\frac{f_j}{2}\right] is odd \end{cases}$$ ([

x] is the integer part of x)

8.2.2. Remarks

- 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 less coefficients than those at scale j.
- From the first remark, ith an initial signal of 2^{Xmax} by 2^{Ymax} points, we deduce that :

. At scale j, there ar ^{j} wavelet packets of 2^{Xmax-j} by 2^{Ymax-j} coefficients, so we still have 2^{Xmax} by 2^{Ymax} coefficients.

. We can then use the following representation of the two-dime sional wavelet packet decomposition.

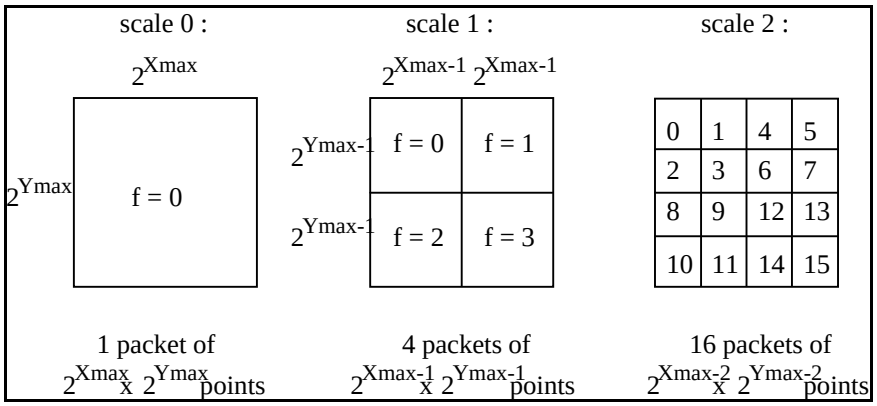

Figure 8.2.2 : Representation of a 2D wavelet packet decomposition

. If Xmax=Ymax=nmax, then at scale nmax, there are 4^{nmax} wavelet packets of 1 coefficient.

8.2.3. General principle of a parallel decomposition algorithm

To do the decomposition in parallel, we chosed to separate each scale (represented by a field of 2^{Xmax} by 2^{Ymax} points) in 4^{jp} ($0 \leq jp < nmax$) parts of equal size. On a fixed process is always stored the same part of the field so we identify the process name by the name of the packet it contains at scale jp. Here is an example of the wavelet packets onto 4 processes (jp = 1) from scale 0 to 2 :

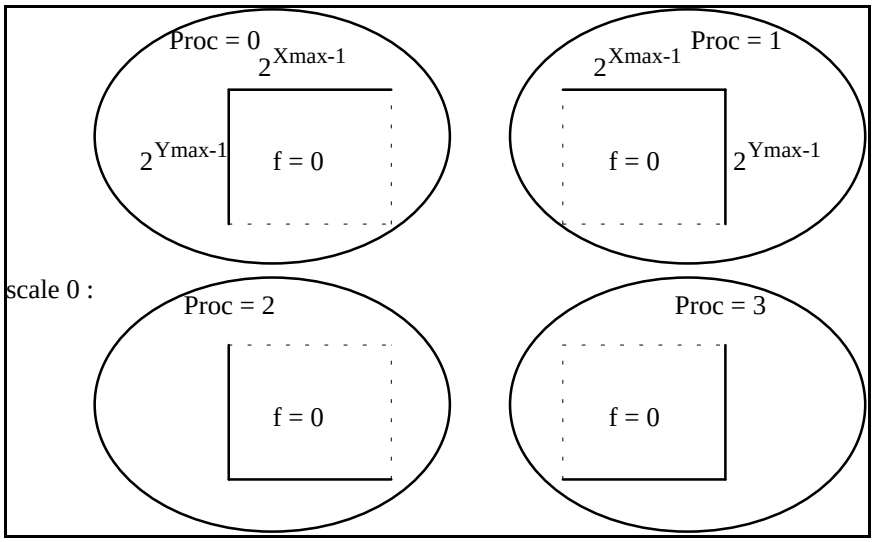

Figure 8.2.3.1 :
Repartition of the wavelet packets of scale 0 onto 4 processes

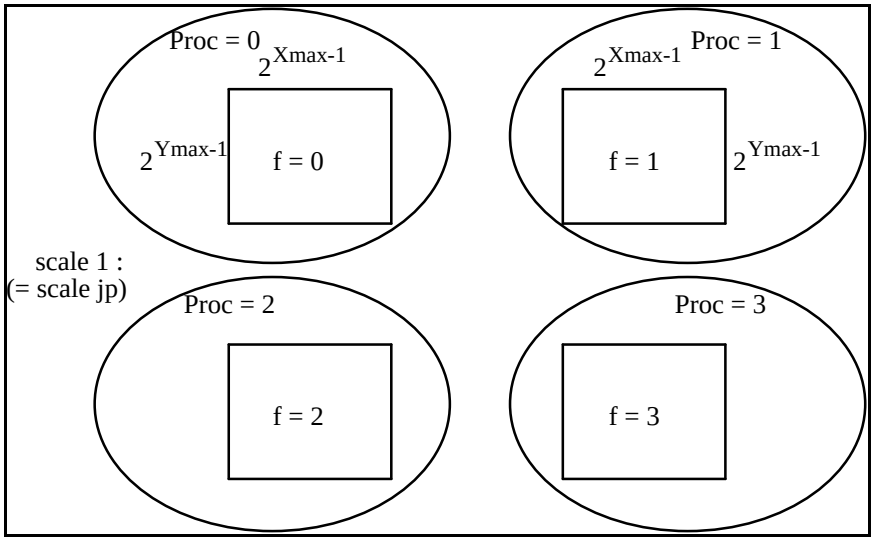

Figure 8.2.3.2 :
Repartition of the wavelet packets of scale 1 onto 4 processes

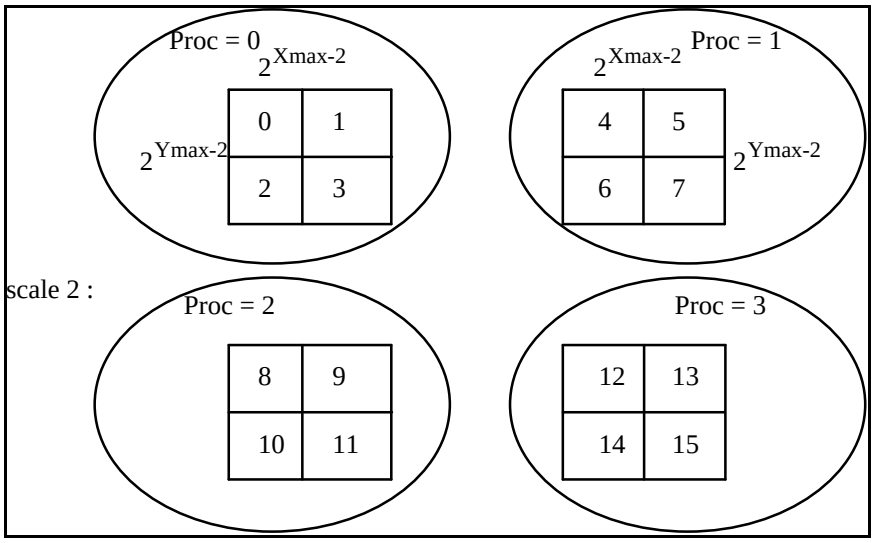

Figure 8.2.3.3 :
Repartition of the wavelet packets of scale 2 onto 4 processes

From the decomposition formula, one sees that the packets at scale j+1 are obtained from those at scale j. Thus, to calculate the decomposition in parallel, using the former field cut, is done in two different ways whether j is smaller o greater than jp.

8.2.4. A parallel decomposition algorithm

First, we consider the case when j < jp

A packet at scale j is shared by 4^{jp-j} processes. In a first step, those processes will exchange their data in order to have the whole shared packet on each of them. Then, in a second step, they will calculate their own part of the packet they share at scale j+1. This part is only the entire 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.

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

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

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

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

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

And 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 what are the processes sharing the packet at scale j=1 and frequency f=2.

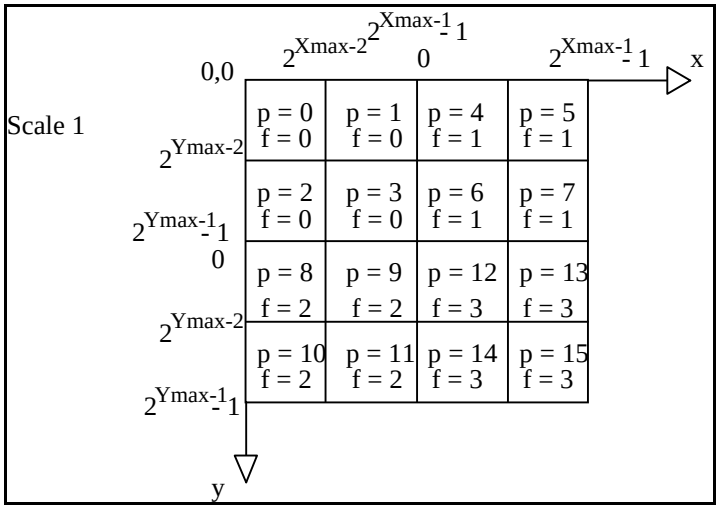

Figure 8.2.4 :
Example of the repartition of wavelet packet coefficients
at scale 1 with 16 processes.

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

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

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

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

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

We will do the transfers iteratively, on the index jj :

- We start with jj equals jp -1, we then define pivot processes named depl, $depl = 4 \cdot \left[\frac{p}{4^{jp-jj}} \right]$. In a first step processes 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 a second step, processes verifying $\left[\frac{p}{4^{jp-jj}} \right] = depl + i$ (C) an processes verifying $\left[\frac{p}{4^{jp-jj}} \right] = depl + 2 + i$ (D) with i equals 0 o 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 decrease jj of one, and do the former point until jj equals j.
- At this point, each process sharing the same packet contains all the data of this packet.

An example is advisable :

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 exchange its data with process 1 (Meantime, process 2 exchange its data with process 3). Then in

the second step process 0 exchange its new set of data (its initial data plus process 1 initial data) with process 2 new set of data. (Meantime process 1 does the same with process 3). We exit the loop on jj, and every processes carry the packet ${}^{0}C^{0}$

Here is the algorithm in pseudo-code of the function doing the data transfers :

Here are the parameters of function *Data transfers*
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 frequenc
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 transfe

Here are its contents

//First initialization
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 transfe
FOR jj = jp-1 TO j BY STEP -1
f_father = f_child / 4
depl = 4 * f_fathe
IF (((p/facto) 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_fathe
factor = 4*factor
ENDFOR j

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

To calculate the transfer complexity, we will assume that the cost of a data transfer is independent of the size of the data set, although it is approximatel 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.

The transfer complexity for the function *Data transfers* is then 4*(jp-j).

We can write the algorithm of the decomposition when j < jp :

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 ${}^{f}C^{j}$

$${}^{f}C^{j} = Datatransfers\left(p, jp, j, f, 2^{Xmax}, 2^{Ymax}, {}^{f}lC^{j}\right)$$

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

//Third, we calculate ${}^{fc}lC^{j+1}$:
base_x = cx(p,jp,width)-cx(f,j,width)
base_y = cy(p,jp,height)-cy(f,j,height)

// Fourth we calculate the local child packet

$${}^{fc}lC_{k,l}^{j+1} = \sum_{kk,ll} FF_{kk,ll} \; {}^{f}C_{kk+2*(base\_x+k),ll+2*(base\_y+l)}^{j}$$

ENDFOR j

In this algorithm, the function *Data transfers* is done jp times, we deduce that its transfer complexity is of ord er O(2*jp*(jp+1)). The computational 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 complexit is O(2*[$\log_4$(NP)]^2) and the calculus complexity is O(P*$\log_4$(NP)*N/NP).

Second the case when j ≥ 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.

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 calculate ${}^{4f+d}C^{j+1}$:

$$ {}^{4f+d}C^{j+1}_{k,l} = \sum_{kk,ll} FF_{kk,ll} \; {}^{f}C^{j}_{kk+2*k,ll+2*l} $$

ENDFOR d
ENDFOR f
ENDFOR j

In this part of the algorithm, there is no transfer complexity and the computational complexit 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 complexity of this part is $O((\log_4(N)-\log_4(NP))*N/NP)$.

Complexities of the parallel decomposition algorithm

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

In a simpler form, 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. A PARALLEL WAVELET PACKET BEST BASIS SELECTION

8.3.1. Definition

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

But if we conside a family of wavelet packets ${}^{f}C^{j}$ *with* $j \in J$ *and* $f \in F_J$, select 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.

The solution we have chosen consists in finding a "best basis". At each packet ${}^{f}C^{j}$, we associate a value $M\left({}^{f}C^{j}\right)$, result of a measure 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 common used measure is Me, the one which evaluates the entropy of a wavelet packet : $\left\{ \begin{array}{l} Me(\varnothing)=0 \\ Me({}^{f}C^{j})=-\sum_{k,l}({}^{f}C^{j}_{k,l})^2 * \ln\left[({}^{f}C^{j}_{k,l})^2\right] \end{array} \right\}$.

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

This is done easily 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 $v_1=\sum_{d=0}^{3}{}^{4*f+d}C^{j+1}_{value}$. 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 do not keep 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 equal ${}^{f}C^{j}_{value}$ to v_1. We decrease j of one and start again the algorithm until j equals 0. At scale 0, all the kept packets form the "best basis" of the signal.

8.3.2. General principle of a parallel best basis selection

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, from scale 0 to scale jp-1, the wavelet packets will be shared by more than one process and from scale jp to nmax, they will be part of a process.

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

packet) a variable 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 $M\left({}^{f}C^{j}\right)$ for every packets of the decomposition. In the next algorithm, we will then consider it as a constant, and therefore, will not contribute to the computational complexity.

8.3.3. A parallel best basis selection algorithm

First, 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 belong to each process.

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

$^{f}C^{j}_{status} = NOT\_KEPT$

$^{f}C^{j}_{value} = v_1$

ENDIF

ENDFOR f

ENDFO j

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

Second, the case when j < jp

From the parallel decomposition, we have seen that at scale j, the process p is storing the local packet $^{\left[\frac{p}{4^{jp-j}}\right]}lC^{j}$. At scale j, we then have that the packet $^{f}C^{j}$ is shared by the processes $p=4^{jp-j}\cdot f$ to $p=4^{jp-j}\cdot(f+1)-1$. Wit another small calculation, one can see that the packets $^{4*f+d}C^{j+1}(d=0..3)$ are shared by exactly the same processes.

From these three remarks, we deduce that if we add $lv_0=M\left(^{\left[\frac{p}{4^{jp-j}}\right]}lC^{j}\right)$ between 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 $v_0=M\left(^{f\_base}C^{j}\right)$ and if we add $^{\left[\frac{p}{4^{jp-j-1}}\right]}lC^{j+1}_{value}$ between 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 all those local values in order to get v_0 and v_1. However, we are twice redundant : we do the same calculation at scale j+1 and at scale j. (once for calculatig v_0 and the second time for v_1). But we can easil 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 ${}^{4\cdot f\_base+d}C^{j+1}$ has the same local value ${}^{4*f\_base+d}lC_{value}^{j+1}$, so the calculation of v_1 as we explained it formerly 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 much 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 : $v_1=\sum_{i=0}^{i=3}\left[\frac{depl+\left(p \bmod 4^{jp-j-1}\right)+4^{jp-j-1}\cdot i}{4^{jp-j-1}}\right]lC_{value}^{j+1}$ wit $depl=4^{jp-j}\cdot f\_base$.

An example is more than advisable :

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

In conclusion, suppose we are on process p, the parallel algorithm becomes :

- We start at j = jp-1.
- First, we calculate $f\_base=\left[\frac{p}{4^{jp-j}}\right]$ and $depl=4^{jp-j}\cdot f\_base$. Second, we calculate $v_0=\sum_{i=0}^{4^{jp-j}-1}M\left(\left[\frac{depl+i}{4^{jp-j}}\right]lC^{j}\right)$ by adding all the values of the local packets at scale j. Third, we calculate $v_1=\sum_{i=0}^{i=3}\left[\frac{depl+\left(p \bmod 4^{jp-j-1}\right)+4^{jp-j-1}\cdot i}{4^{jp-j-1}}\right]lC_{value}^{j+1}$ 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 $\left[\frac{p}{4^{jp-j}}\right]lC^{j}$, we equal $\left[\frac{p}{4^{jp-j}}\right]lC_{value}^{j}$ to v_0, and we do not keep all its descendants. If v_0 is greater than v_1, we keep the local packet $\left[\frac{p}{4^{jp-j-1}}\right]lC^{j+1}$, and we equal $\left[\frac{p}{4^{jp-j}}\right]lC_{value}^{j}$ to v_1.
- We decrease j of one and do the former point until j equals 0.

One solution to exchange the local values consists in taking all the p rocesses involved and exchange their local values two by two, until every processes have the total sum. The next figure explains graphically how it works.

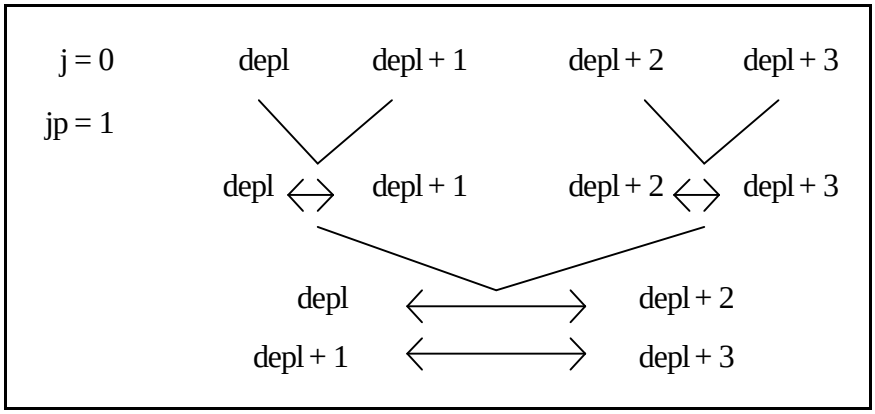

Figure 8.3.4.1 :
Transfer of the local values v_0 between processes depl to depl+3

To calculate v_1, we will use the same principle, except that the first transfe may not be anymore between process depl and depl+1 but will be between depl and depl+4^{jp-j-1}.

Here is then the algorithm of the function doing these transfers :

Here are the parameters of function *Local value transfer* :
- p is the name of the current process
- np is the number of processes doing the transfe
- p_base is the pivot process
- p_shift is the shift of processes to do the transfe
- $local\_value_p$ is the value of the process p to transfe

Here are its contents :

```
test=np
stepsize=p_shift
pvalue=local_value_p

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 transfer complexity of this function is of order $O(2*\log_2(test/p\_shift))$ and the computation complexity is of order $O(\log_2(test/p\_shift))$.

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)

    v0 = Local value transfer(p,4^(jp-j),f,1, M( f lC^j ))
    v1 = Local value transfer(p,4^(jp-j),f,4^(jp-j-1), f_child lC_value^(j+1) )

    IF (v0 ≤ v1)
        f lC_status^j = KEPT
        f lC_value^j = v0
        factor=1
```

FOR jj=j+1 TO nmax
factor = factor*4
f_base = factor*f
FOR ff = 0 TO facto
$^{f\_base+ff}lC^{jj}_{status} = NOT\_KEPT$
ENDFOR ff
ENDFOR jj
ELSE
$^{f}lC^{j}_{status} = NOT\_KEPT$
$^{f}lC^{j}_{value} = v_1$
ENDIF
ENDFOR j

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

Complexities of the parallel best basis selection algorithm

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

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

8.4. A PARALLEL WAVELET PACKET RECONSTRUCTION

8.4.1. Definition

The reconstruction is the opposite of the decomposition, from a family of kept wavelet packets $^{f_j}C^{j}$ wit $j \in J$ (where $J \subset \mathrm{N}_{nmax} = \{n \in \mathrm{N}; 0 \le n \le nmax\}$) and $f_j \in F_J$ (where for each $j \in J$, $F_j \subset \mathrm{N}_{4^j-1}$), we want to obtain

$s_{k,l} = {}^0C^0_{k,l}$ (where $0 \le k < 2^{Xmax}$ and $0 \le l < 2^{Ymax}$) the corresponding signal representation (S). We obtain S by completing the following recursive formula until we get ${}^0C^0$:

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

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 directly this formula, rather we reconstruct a whole branch of the tree, one level at a time. Instead of having its entire fathe $\left[\frac{f_j}{4}\right] C^{j-1}$, we only have one p art 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 are all zeros) if it is not a kept packet.

8.4.2. General principle of a parallel reconstruction algorithm

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

8.4.3. A parallel reconstruction algorithm

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

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

Second, the case when scale j ≤ jp

A first solution for reconstructing is to consider that each process has an entire data set. Each of them will do locally the reconstruction in sequential and when they reach scale 0, each of them has a partial signal. We need then to add all these partial signals in order to get the whole signal. Unfortunately, even if w 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 used less memory

than in sequential. The solution we propose will have a lower calculation complexity, but we will increase the number of transfers.

At scale j ($1 \leq j \leq jp$), we reconstruct the local packet $^{f=\left[\frac{p}{4^{jp-j}}\right]}lC^{j}$ in a real size temporary father packet $^{f/4}tC^{j-1}$ by the following formula : $^{f/4}tC_{k,l}^{j-1} = {}^{f/4}tC_{k,l}^{j-1} + \sum_{n,m} FF_{n-2(base\_x+k),m-2*(base\_y+l)} * {}^{f}lC_{n,m}^{j}$ where FF is the filter F(f mod 4,f / 4), and base_x and base_y are the x - and y- shifts to have the real position 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 $^{f/4}C^{j-1}$ exchange and add 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 decrease j of one and start again 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 $0 \leq j0 \leq jp$) on at least one process. We handle this problem in a very simple manner, if the packet $^{p}lC^{jp}$ does not exist, we create it artificially to the null packet. The propagation of this packet in all scales j smaller than jp will handle the problem once and fo all.

In a first part, we study the function we use to transfer all the temporary packets between the concerned processes, then we explain the algorithm for the reconstruction.

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

Here are the parameters of function *Local array transfer* :

p is the name of the current process
np is the number of processes doing the transfe
p_base is the pivot process
p_shift is the shift of processes to do the transfe
$local\_array_p$ is the array of the process p to transfe
size_array is the size of the local array

Here are its contents :

test=np
stepsize=p_shift
pvalue[i]=local_value$_p$[i] for $0 \leq i < size\_array$

```
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≤i<size _ array

    test = test/2
    stepsize = stepsize * 2
ENDWHILE

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

The transfer complexity of this function is of order O(2*$\log_2$(test/p_shift)) and the computational complexity is of order O(size_array*$\log_2$(test/p_shift)).

Here is then the reconstruction algorithm :

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
    $bx = cx\left(p, jp, 2^{Xmax}\right) - cx(f2, j-1, 2^{Xmax})$
    $by = cy\left(p, jp, 2^{Ymax}\right) - cy(f2, j-1, 2^{Ymax})$
    $base\_x = cx\left(p, jp, 2^{Xmax}\right) - cx(f, j, 2^{Xmax})$
    $base\_y = cy\left(p, jp, 2^{Ymax}\right) - cy(f, j, 2^{Ymax})$

    2/ We create a real size temporary fathe ${}^{f2}tC^{j-1}$
    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 $= 2^{Xmax-j+1} * 2^{Xmax-j+1}$

$${}^{f2}tC^{j-1} = Local\ array\ transfer\left(p, 4^{jp-j+1}, f2 * 4^{jp-j+1}, 1, {}^{f2}tC^{j-1}, len\right)$$

6/ If ${}^{f2}lC^{j-1}$ does 't exist we set it to the null vecto

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

ENDFO

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 $O(jp*2^{Xmax}*2^{Ymax}/4^{jp})$.
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 $O(\log_4(NP)*N/NP)$.

Complexities

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

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

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

8.5.1. Definition

We try 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).

In a first part we decompose the matrix C in a wavelet packet best basis, we obtain a family of kept wavelet packets ${}^{cf_j}C^j$ wit $j \in J$ (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 a second part, we decompose the vector D in all the possible one dimensional wavelet packets ${}^{df_j}D^j$ wit $0 \le j \le nmax$ and $0 \le df_j < 2^j$.

Each frequency cf_j is the combinaison of a frequency cf_x_j among 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 :

$$cf\_x_j = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left(\left[\frac{cf_j}{4^{j-jj-1}} \right] \bmod 2 \right)$$

$$cf\_y_j = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left(\left[\frac{\left[\frac{cf_j}{4^{j-jj-1}} \right]}{2} \right] \bmod 2 \right)$$

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

Then, we reconstruct all the packets ${}^{cf\_y_j}E^j$ to obtain the vector E.

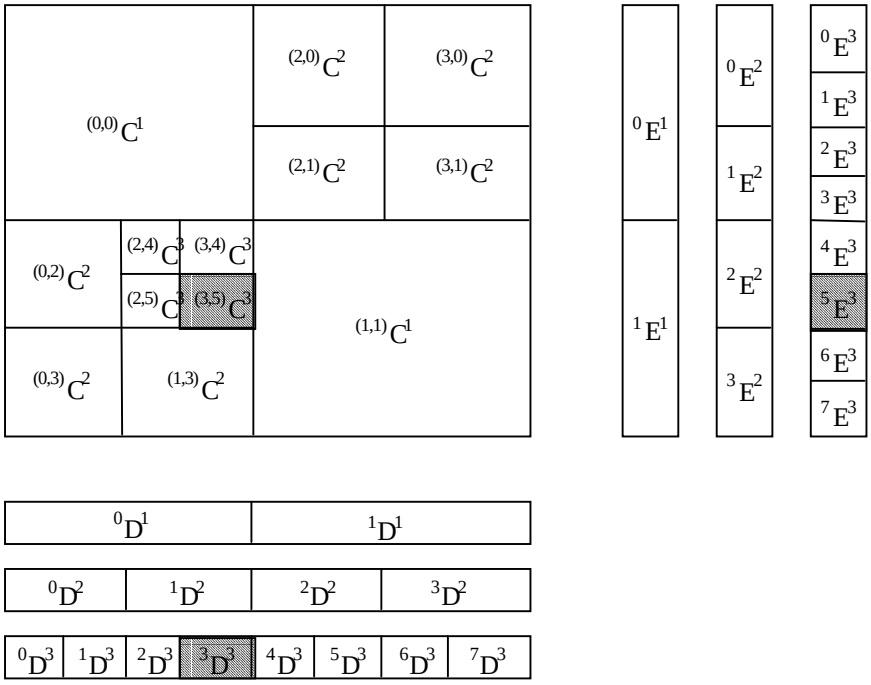

Figure 8.5.1.1 :
Example of the multiplication in the wavelet packet expansion,
we consider ${}^{5}E^{3} = {}^{(3,5)}C^{3} \times {}^{3}D^{3}$

8.5.2. General principle of a parallel multiplication algorithm

We can use the former parallel algorithms in order to perform in a simple manner the multiplication of a matrix by a vector.

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

The multiplication between the wavelet packets can now take place :

- 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 ${}^{fx}lD^{j}$), we 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 reconstruct all the kept packets to get their partial vector E.

In a last part, each process exchange and add all the partial vectors E in orde to get the whole one. E is the result of the multiplication of C by D.

8.5.3. Algorithm of the parallel multiplication

Here is this algorithm :

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

$$base\_x = cx(p, jp, 2^{Xmax}) - cx(f, j, 2^{Xmax})$$
$$base\_y = cy(p, jp, 2^{Ymax}) - cy(f, j, 2^{Ymax})$$

2/ We set the frequencies among the x- and y- axis

$$fx = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left(\left\lfloor \frac{f}{4^{j-jj-1}} \right\rfloor \bmod 2 \right)$$

$$fy = \sum_{jj=0}^{j-1} 2^{j-jj-1} \cdot \left(\left\lfloor \frac{\left\lfloor \frac{f}{4^{j-jj-1}} \right\rfloor}{2} \right\rfloor \bmod 2 \right)$$

3/ We keep the local vecto

$${}^{fx}lD^{j}_{0..wh-1} = {}^{fx}D^{j}_{base\_x+0..wh-1}$$

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 smalle
                    matrix-vector multiplication
                    ^fy E^j = ^(f=(fx,fy)) C^j × ^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 all the partial vector E to have the whole vector E

$$E=Localarraytransfer\left(p,4^{jp},0,1,E,2^{Ymax}\right)$$

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

8.5.4. Complexities

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 t ansfer.

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(^{2}), and if R << N then this method is very efficient.

8.6. CONCLUSION