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 \( 0 \leq i < 2^{X_{\text{max}}} \) and \( 0 \leq j < 2^{Y_{\text{max}}} \), we pose \( n_{\text{max}} = \min(X_{\text{max}},Y_{\text{max}}) \). In a first step we want to
calculate all the wavelet packets $f_j C^j$, sets of wavelet packet coefficients $f_j C_{k_j,l_j}^j$ defined as follows:

- The indices are in the range:

$$0 \leq j \leq n_{\text{max}}$$
$$0 \leq k_j < 2^{X_{\text{max}}-j}$$
$$0 \leq l_j < 2^{Y_{\text{max}}-j}$$
$$0 \leq f_j < 4^j$$

with $j,k,l$ and $f \in \mathbb{N}$

- The initial packet is:

$$0^0 C_{k,0}^0 = s_{k,1} \text{ with } 0 \leq k < 2^{X_{\text{max}}} , 0 \leq l < 2^{Y_{\text{max}}}$$

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

$$f_{j+1}^d = 4^{j+1} C_{k_j+l_j}^{j+1} = \sum_{n_j,m_j} F(d,f_j)_{n_j-2k_j+l_j-2l_j} + f_j C_{n_j,m_j}^j \quad \text{with} \quad d = 0, 1, 2 \text{ or } 3$$

We shall say that the packet $f_j C^j$ is the father of the four packets $4^{j+1} C^{j+1}$ or that they are its four children.

- From a one-dimensional wavelet $\Psi$ (defined by its filter $G$) and its smoothing function $\Phi$ (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 $F(d,f_j)_{n,m} = \text{Filter(gray code}_2 d (d,f_j))_{n,m}$ where:

$$\text{gray code}_2 d (d,f_j) = 2 \cdot a + b$$

with:

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

and:

$$b = \begin{cases} \left \lfloor \frac{d}{2} \right \rfloor & \text{if } f_j \text{ is even} \\ \left \lfloor \frac{f_j}{2} \right \rfloor & \text{if } f_j \text{ is odd} \end{cases}$$

(\lfloor x \rfloor \text{ 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, an initial signal of $2^{X_{\text{max}}}$ by $2^{Y_{\text{max}}}$ points, we deduce that:

$$\sum_{j=0}^{n_{\text{max}}} 4^j C_{k,j}^j$$
At scale $j$, there are $j$ wavelet packets of $2^{X_{\text{max}}-j}$ by $2^{Y_{\text{max}}-j}$ coefficients, so we still have $2^{X_{\text{max}}}$ by $2^{Y_{\text{max}}}$ coefficients.

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

![Wavelet Packet Decomposition Diagram](image)

**Figure 8.2.2**: Representation of a 2D wavelet packet decomposition

If $X_{\text{max}}=Y_{\text{max}}=n_{\text{max}}$, then at scale $n_{\text{max}}$, there are $4^{n_{\text{max}}}$ wavelet packets of 1 coefficient.

### 8.2.3. General principle of a parallel decomposition algorithm

To do the decomposition in parallel, we chose to separate each scale (represented by a field of $2^{X_{\text{max}}}$ by $2^{Y_{\text{max}}}$ points) in $4^{j_{p}}$ $(0 \leq j_{p} < n_{\text{max}})$ 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 $j_{p}$. Here is an example of the wavelet packets onto 4 processes ($j_{p} = 1$) 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
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 or greater than \( j_p \).

### 8.2.4. A parallel decomposition algorithm

First, we consider the case when \( j < j_p \)

A packet at scale \( j \) is shared by \( 4^{j_p-j} \) 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 \( j_p \).

From the decomposition formula, we obtain that at scale \( j \), the frequency \( f \) of the packet stored on process \( p \) is given by

\[
    f = \left\lfloor \frac{P}{4^{j_p-j}} \right\rfloor
\]

where \([x]\) is the integer part of \( x \). Reciprocally, a packet at scale \( j \) and frequency \( f \) is shared by the processes \( p = 4^{j_p-j} \cdot f \) to \( p = 4^{j_p-j} \cdot (f+1) - 1 \). 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:

- \( \text{base}_x = cx(p, jp, 2X_{\text{max}}) - cx \left( \left\lceil \frac{p}{4^{jp-j}} \right\rceil, 2X_{\text{max}} \right) \) where

\[
cx(p, jp, 2X_{\text{max}}) = \begin{cases} 0 & \text{if } jp = 0 \\ \sum_{m=1}^{jp} 2X_{\text{max}} - m, & \text{otherwise} \end{cases}
\]

- \( \text{base}_y = cy(p, jp, 2Y_{\text{max}}) - cy \left( \left\lceil \frac{p}{4^{jp-j}} \right\rceil, 2Y_{\text{max}} \right) \) where

\[
cy(p, jp, 2Y_{\text{max}}) = \begin{cases} 0 & \text{if } jp = 0 \\ \sum_{m=1}^{jp} 2Y_{\text{max}} - m, & \text{otherwise} \end{cases}
\]

And the width and height of the part of the packet stored on a process at scale j is \( 2X_{\text{max}} - 2X_{\text{max}} \) by \( 2Y_{\text{max}} - 2Y_{\text{max}} \).

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.

![Scale Diagram]
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 \( ^{2}C_{0}^{1} \) are processes 8 to 11. Each of them have 
\( \text{wh} = 2^{\text{max} \cdot jp} \) by \( \text{ht} = 2^{\text{max} \cdot jp} \) data.

- Process 8 contains the local packet \( ^{2}lC_{0,\.wh-1,0,.ht-1}^{1} = ^{2}C_{0,\.wh,0,.ht-1}^{1} \)
- Process 9 contains the local packet \( ^{2}lC_{0,\.wh-1,0,.ht-1}^{1} = ^{2}C_{0,.wh,2\.wh-1,0,.ht-1}^{1} \)
- Process 10 contains the local packet \( ^{2}lC_{0,.wh-1,0,.ht-1}^{1} = ^{2}C_{0,.wh,.ht,.2\.ht-1}^{1} \)
- Process 11 contains the local packet \( ^{2}lC_{0,.wh-1,0,.ht-1}^{1} = ^{2}C_{0,.wh,.2\.wh-1,\.ht,.2\.ht-1}^{1} \)

We will do the transfers iteratively, on the index \( jj \):
- We start with \( jj \) equals \( jp - 1 \), we then define pivot processes named \( \text{depl} \), \( \text{depl} = 4 \left[ \frac{p}{4^{jp-\text{jj}}} \right] \). In a first step processes verifying

\[
\left[ \frac{p}{4^{jp-\text{jj}}} \right] = \text{depl} + i \quad (\text{A})
\]

exchange their data with processes

\[
\left[ \frac{p}{4^{jp-\text{jj}}} \right] = \text{depl} + i + 1 \quad (\text{B})
\]

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

\[
\left[ \frac{p}{4^{jp-\text{jj}}} \right] = \text{depl} + i \quad (\text{C})
\]

processes verifying

\[
\left[ \frac{p}{4^{jp-\text{jj}}} \right] = \text{depl} + 2 + i \quad (\text{D})
\]

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 $^0C^0$

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 $4^p$ 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

**Here are its contents**

//First initialization
f = p / $4^p$
factor = 1
f_child = p
llx = width / $2^p$
llly = height / $2^p$
base_x = cx(p,jp,width)-cx(f,j,width)
base_y = cy(p,jp,height)-cy(f,j,height)

$^fC^i_{base_x+0..wl-1,base_y+0..hl-1} = Local\_packet_{0..w-1,0..h-1}$

//Then transfer
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 $^fC^i_{bx+0..lx-1,by+0..ly-1}$ 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 $^fC^i_{bx+0..lx-1,by+0..ly-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ 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 $fC^j_{bx+0,Jx-1,by+0,Jy-1}$ to process p-2*factor
ENDIF

lly = lly*2
f_child = f_fathe
factor = 4*factor
ENDFOR j

Data transfers returns the packet $fC^j$. 

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 approximate 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 \textit{Data transfers} is then $4^*(jp-j)$.

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

\[
p \text{ is the process name} \\
4^p \text{ is the number of processes} \\
w_h = \text{width } / 2^p \\
h_t = \text{height } / 2^p \\
\]

\[
\text{FOR } j = 0 \text{ TO } jp-1 \\
f = p / 4^{jp-j} \\
f_c = p / 4^{jp-j-1} \\
d = f_c \text{ modulo } 4 \\
\]

\[
//\text{First we exchange data to have } fC^j \\
\text{ }C^j = \text{Data transfers}(p, jp, j, f, 2^{X_{\text{max}}}, 2^{Y_{\text{max}}}, fC^j). \\
\]

\[
//\text{Second, we define the filter to use:} \\
\text{FF} = F(d, f). \\
\]

\[
//\text{Third, we calculate } fC^{j+1}: \\
\text{base}_x = \text{cx}(p, jp, \text{width}) - \text{cx}(f, j, \text{width}) \\
\text{base}_y = \text{cy}(p, jp, \text{height}) - \text{cy}(f, j, \text{height}) \\
\]

\[
//\text{Fourth we calculate the local child packet} \\
fC^{j+1} = \sum_{kk, ll} FF_{kk, ll} fC^j_{kk + 2^*(\text{base}_x + k), ll + 2^*(\text{base}_y + l)} \\
\]

ENDFOR

In this algorithm, the function \textit{Data transfers} is done $jp$ times, we deduce that its transfer complexity is of order $O(2*jp*(jp+1))$. The computational complexity is of order $O((jp*2^{X_{\text{max}}} + 2^{Y_{\text{max}}}) / 4^p)$ where the constant depends on the length of the filters. In simple terms, if $NP=4^p$ is the number of processes and if we have $N=4^{n_{\text{max}}}$ points ($X_{\text{max}}=Y_{\text{max}}=n_{\text{max}}$), then the transfer complexity is $O(2^*(\log_4(NP))^2)$ and the calculus complexity is $O(P*\log_4(NP)*N/NP)$.

\textbf{Second the case when } j \geq jp

Each process has $4^{j+1}p$ 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 = 4jp * p TO 4jp * (p+1) - 1
        FOR d = 0 TO 3
        First we set the filter to use : FF = F(d,f)
        The we calculate $\sum_{kk,ll} 4f+d C_{kk,ll}^{j+1} = FF C_{kk+2^s,j+2^s}^j$
        ENDFOR d
    ENDFOR f
ENDFOR j

In this part of the algorithm, there is no transfer complexity and the computational complexit is of order $O((n_{max} - jp)*2^{X_{max}}*2^{Y_{max}}/4^p)$. If we consider $NP=4^p$ processes and $N=4^{n_{max}}$ 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 parallel decomposition algorithm, we conclude the transfer complexity is of order $O(2^{*}jp*(jp+1))$ and the computational complexity is of order $O(n_{max}*2^{X_{max}}*2^{Y_{max}}/4^p)$.

In a simpler form, if $N=4^{n_{max}}$ is the number of points ($X_{max}=Y_{max}=n_{max}$) and $NP=4^p$ is the number of processes then the transfer complexity is $O(2^[log_4(NP)])$ 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 consider 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 \(4^j f_{0} + d_{0} C_{j,0}^{j+1}\), \(4^j f_{0} + d_{1} + d_{0} C_{j,1}^{j+2}\), 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 \(M_e\), the one which evaluates the entropy of a wavelet packet:

\[
M(\emptyset) = 0 \\
M(f C^j) = -\sum_{k,j} (f C_{k,j}^j)^2 \ln(f C_{k,j}^j)^2.
\]

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

This is done easily by an iterative algorithm. We define \(f C_{value}^j\) 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 \(j = \text{max}\), we initialize \(f C_{value}^j = M\left( f C^j \right)\). Then we start the algorithm at scale \(j = \text{max} - 1\). For all the frequencies \(f\) at scale \(j\), we compare \(v_0 = M\left( f C^j \right)\) to \(v_1 = \sum_{d=0}^{3} 4^j f_d + d_1 C_{value}^{j+1}\). If \(v_0\)

is smaller than or equal to \(v_1\) we keep the wavelet packet \(f C^j\), we set \(f C_{value}^j = v_0\), and we do not keep all its descendants. If \(v_0\) is greater than \(v_1\), we keep the four children packets \(4^j f + d C_{j+1}\), and we equal \(f C_{value}^j\) 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 \(4^p\) processes, from scale 0 to scale \(j p - 1\), the wavelet packets will be shared by more than one process and from scale \(j p\) to \(\text{max}\), 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 \(j p\). In order to know if a packet is kept or not, we define \(f C_{status}^j\) (\(f C_{status}^j\) for a local
packet) a variable equal to $KEPT$ if the wavelet packet $^j C^i$ 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 $M \left( ^j C^i \right)$ 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 $j \geq j_0$**

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.

```plaintext
FOR f = $4^{n_{\text{max}}-jp} \ast p$ TO $4^{n_{\text{max}}-jp} \ast (p+1) - 1$
    $^f C_{\text{status}} = KEPT$
    $^f C_{\text{value}} = M \left( ^f C^i \right)$
ENDFOR f

FOR $j = n_{\text{max}}-1$ TO $jp$ BY STEP -1
    FOR f = $4^{jp} \ast p$ TO $4^{jp} \ast (p+1) - 1$
        $v_1 = \sum_{d=0}^{4^{f+1}} C_{\text{value}}^{j+1}$
        $v_0 = M \left( ^f C^i \right)$
        IF ($v_0 \leq v_1$)
            $^f C_{\text{status}} = KEPT$
            $^f C_{\text{value}} = v_0$
            factor = 1
        ENDIF
    ENDFOR f
ENDFOR j

ELSE
```

```plaintext
factor = factor \ast 4
f_base = factor \ast f
FOR ff = 0 TO facto
    $^f C_{\text{status}} = NOT \_ KEPT$
ENDFOR ff
ENDFOR jj
```

ELSE
\[ f \text{ C}_{\text{status}} = \text{NOT \_KEPT} \]

\[ f \text{ C}_{\text{value}} = v_1 \]

ENDIF
ENDFOR f
ENDFOR j

The computational complexity of this algorithm is of order \(O(4/3^*(4^{\text{max}}\cdot j p - 1))\). Therefore, if we have \(N P=4^{j p}\) processes and \(N=4^{\text{max}}\) points, this complexity 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 \(\left[ \frac{p}{4^{j p-j}} \right] lC^j\). At scale \(j\), we then have that the packet \(f \text{ C}^j\) is shared by the processes \(p=4^{lp-j}, f\) to \(p=4^{lp-j}(f+1)-1\). With another small calculation, one can see that the packets \(4^{f+d} C^{j+1}(d=0..3)\) are shared by exactly the same processes.

From these three remarks, we deduce that if we add \(lv_0=M\left(\left[ \frac{p}{4^{j p-j}} \right] lC^j\right)\) between processes \(p=4^{lp-j}, f\_base\) to \(p=4^{lp-j}(f\_base+1)-1\) where \(f\_base=\left[ \frac{p}{4^{j p-j}} \right]\), we will obtain \(v_0=M\left(f\_base C^j\right)\) and if we add \(\left[ \frac{p}{4^{j p-j+1}} \right] lC_{value}^{j+1}\) between the same processes, we will get

\[ v_1=\sum_{d=0}^{3} 4^{j f\_base+d} C_{value}^{j+1}. \]

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 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 \(\left[ \frac{p}{4^{j p-j}} \right] lC^j\) to the winner.
Then, each process sharing $4\cdot f_{\text{base}}d\cdot C^{j+1}$ has the same local value $4\cdot f_{\text{base}}d\cdot IC^{j+1}$, 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: 

$$v_1 = \sum_{i=0}^{j-1} \left[\sum_{i=0}^{\text{depl}\left(\text{mod } 4^{j+1}\cdot j\right)+4^{j+1}} 4^{j+1}\cdot IC_{\text{value}}^{j+1}\right]$$ 

with $\text{depl} = 4^{jp-j} \cdot f_{\text{base}}$.

An example is more than advisable:

Suppose that $jp=2$, $j=0$ and $p=5$. To calculate $v_1$, the previous formula gives $v_1 = 4^{j+1} IC_{\text{value}}^{j+1} + 5 IC_{\text{value}}^{j+1} + 9 IC_{\text{value}}^{j+1} + 13 IC_{\text{value}}^{j+1}$ 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 $f_{\text{base}} = \left[\frac{p}{4^{jp-j}}\right]$ and $\text{depl} = 4^{jp-j} \cdot f_{\text{base}}$.
- Second, we calculate $v_0 = \sum_{i=0}^{4^{tp-j}-1} M \left[\sum_{i=0}^{\text{depl}\left(\text{mod } 4^{j+1}\cdot j+4^{j+1}\cdot i\right)} 4^{j+1} \cdot IC_{\text{value}}^{j+1}\right]$ by adding all the values of the local packets at scale $j$. Third, we calculate $v_1 = \sum_{i=0}^{j-1} \left[\sum_{i=0}^{\text{depl}\left(\text{mod } 4^{j+1}\cdot j+4^{j+1}\cdot i\right)+4^{j+1}} 4^{j+1}\cdot IC_{\text{value}}^{j+1}\right]$ 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 $\left[\frac{p}{4^{jp-j}}\right] IC_{\text{value}}^{j}$, we equal $\left[\frac{p}{4^{jp-j}}\right] IC_{\text{value}}^{j}$ to $v_0$ and we do not keep all its descendants. If $v_0$ is greater than $v_1$, we keep the local packet $\left[\frac{p}{4^{jp-j}}\right] IC_{\text{value}}^{j+1}$, and we equal $\left[\frac{p}{4^{jp-j}}\right] IC_{\text{value}}^{j+1}$ 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 processes involved and exchange their local values two by two, until every processes have the total sum. The next figure explains graphically how it works.

![Diagram](image)

**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 transfer may not be anymore between process depl and depl+1 but will be between depl and depl+4$^{\text{p-1}}$.

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 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

Here are its contents:
test=np
steps=|p_shift|
pvalue=local_value_p

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

ELSE
    p2=p-stepsiz e
    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
    stepsiz e = stepsiz e * 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\(^p\) is the maximum number of processes

FOR j = \(0\) TO \(4^p\) BY STEP -1
    f = p / \(4^p\)
    f_child = p / \(4^p\)
    \(v_0\) = Local value transfer(p,\(4^p\),f,1,\(M(\text{lC}^j)\))
    \(v_1\) = Local value transfer(p,\(4^p\),f,\(4^p\),f_child,\(\text{lC}^j\)value + \(\text{value}\))
IF (\(v_0\) ≤ \(v_1\))
    /\(\text{lC}^j\)status =KEPT
    /\(\text{lC}^j\)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 \( I_C^{jj} \) = NOT _ KEPT
  ENDFOR ff
ENDFOR jj
ELSE
  \( I_C^{jj} \) = NOT _ KEPT
  \( I_C^{jj} \) = v_1
ENDIF
ENDFOR j

Since we do the external loop \( j \)p times and we call twice the function \( Local value transfer \), the first time with test/p_shift equaling \( 4^{jp} \) and the second time with test/p_shift equaling 4, we deduce that the transfer complexity is of order \( O(2^{jp*jp}) \) and the computational complexity is half of it, \( O(jp*jp) \). If \( NP=4^{jp} \) is the number of processes, then the transfer complexity is of order \( O(2^{j}[\log_4(NP)]^2) \) 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 order \( O(2^{jp*jp}) \) and the computational complexity is of order \( O(4/3*(4^{max*jp} - 1) + jp^{jp}) \).

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

**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 \( \{ J_j \} \) wit \( j \in J \) (where \( J \subseteq \{ n \in \mathbb{N} : 0 \leq n \leq n_{max} \} \)) and \( f_j \in F_j \) (where for each \( j \in J \), \( F_j \subseteq \mathbb{N}_{4^j} \)), we want to obtain
$s_{k,l} = 0 C^{0}_{k,l}$ (where $0 \leq k < 2^{X_{\text{max}}}$ and $0 \leq l < 2^{Y_{\text{max}}}$) the corresponding signal representation (S). We obtain S by completing the following recursive formula until we get $0 C^{0}$:

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

where $F(d, f_j)_{n,m} = \text{Filter} (\text{gray code} 2d(d, f_j))_{n,m}$ 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 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:

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

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$p$ processes, and that the data are dispatched on each of them as before. In the same way that the sequential algorithm, we will start at scale $n_{\text{max}}$ and reconstruct each kept packet until we reach scale 0. There will be again two different algorithms, one when scale $j$ is greater than $p$ and the other one when scale $j$ is smaller than or equal to $p$.

### 8.4.3. A parallel reconstruction algorithm

First, we consider the case when $j > p$

In this case, the algorithm is the same as the sequential one:

\[ p \text{ is the process name} \]

```
FOR j = n_{\text{max}} TO $p$+1 BY STEP -1
```
FOR $f = 4^j p + 1 \* p$ TO $4^j p^* (p+1) - 1$

IF ($\tilde{C}^{j-1}_{\text{not kept}}$ exists)

IF ($\tilde{C}^{j-1}_{\text{kept}}$ = NOT _KEPT_)

We set $\tilde{C}^{j-1}_{k,l} = 0, \forall k,l$

ENDIF

ENDIF

ENDFOR $f$

FOR $f = 4^j p + 1 \* p$ TO $4^j p^* (p+1) - 1$

IF ($\tilde{C}^j$ exists)

IF ($\tilde{C}^j_{\text{kept}}$ = KEPT )

1/ We set the filter to use for reconstructing:
   \[ FF = F(f \mod 4, f/4) \]

2/ We create $\tilde{C}^{j-1}/4$ is it doesn't exist
   by setting all its coefficients to zero.

3/ We set $\tilde{C}^{j-1}/4_{\text{kept}}$ = KEPT

4/ We compute the partial reconstruction:
   $\tilde{C}^{j-1}_{k,l} = \tilde{C}^{j-1}_{k,l} + \sum_{n,m} FF_{k-2n,l-2m} \* \tilde{C}^{j}_{n,m}$

5/ We don't need any more $\tilde{C}^j$

ENDIF

ENDIF

ENDFOR $f$

ENDFOR $j$

In the worst case, the best basis selected is composed of all the packets at scale \( n_{\text{max}} \). In that case, the computational complexity is of order \( O((n_{\text{max}} - j p)^2 X_{\text{max}} Y_{\text{max}}/4^p) \). In any other cases it is less than this complexity. If we have \( N = 4^{n_{\text{max}}} \) points and \( NP = 4^p \) processes, the computational complexity 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 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 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 \( j (1 \leq j \leq \text{jp}) \), we reconstruct the local packet \( f \left( \left[ \frac{p}{4^{w-j}} \right] \right) lC^{j} \) in a real size temporary father packet \( f \left( \frac{4^{w-j}}{4} \right) lC^{j-1} \) by the following formula:

\[
\begin{align*}
    f \left( \frac{4^{w-j}}{4} \right) C^{j-1}_{k,l} &= f \left( \frac{4^{w-j}}{4} \right) C^{j-1}_{k,j-1} + \sum_{n,m} \text{FF}_{n-2(base_{x}+k),m-2^*(base_{y}+l)} \ast f \left( \frac{4^{w-j}}{4} \right) lC^{j}_{n,m}
\end{align*}
\]

where FF is the filter \( F(f \text{ mod } 4, f / 4) \), and \( \text{base}_{x} \) and \( \text{base}_{y} \) are the x- and y-shifts to have the real position of the local packet \( f \left( \frac{4^{w-j}}{4} \right) lC^{j} \). At this point we must add the local packet \( f \left( \frac{4^{w-j}}{4} \right) lC^{j-1} \) to this temporary packet if it is a kept packet. Then, each process sharing the packet \( f \left( \frac{4^{w-j}}{4} \right) C^{j-1} \) 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 \( f \left( \frac{4^{w-j}}{4} \right) lC^{j-1} \). Then we decrease \( j \) of one and start again until \( j = 0 \).

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 \( j_0 \) (with \( 0 \leq j_0 \leq \text{jp} \)) on at least one process. We handle this problem in a very simple manner, if the packet \( \frac{p}{4} lC^{jp} \) does not exist, we create it artificially to the null packet. The propagation of this packet in all scales \( j \) smaller than \( \text{jp} \) will handle the problem once and for 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 \textit{Local array transfer}:
- \( p \) is the name of the current process
- \( n_p \) is the number of processes doing the transfer
- \( p\_\text{base} \) is the pivot process
- \( p\_\text{shift} \) is the shift of processes to do the transfer
- \( \text{local}\_\text{array}_p \) is the array of the process \( p \) to transfer
- \( \text{size}\_\text{array} \) is the size of the local array
Here are its contents:

test=np
stepsze=p_shift
pvalue[i]=local_value_{p[i]} for 0\leq i< size _array

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

total_value[i] = pvalue[i]
Local array transfer returns the array 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^{X_{max}}*2^{Y_{max}} is the size of the field at scale 0
wh = 2^{X_{max}} \cdot j \cdot p
ht = 2^{Y_{max}} \cdot j \cdot p

FOR j = jp TO 1 BY STEP -1
np = 4^p - j
f = p / 4^p - j
f2 = f / 4

IF (f IC j exists)
    IF (f IC status = NOT _ KEPT )
        1/ We set f IC_{k,l} = 0, \forall 0 \leq k < wh, 0 \leq l < ht
        2/ We set f IC_{status} = KEPT
    ENDIF
ELSE
    1/ We create f IC j by setting all its coefficients to zero
    2/ We set f IC_{status} = KEPT
ENDIF

IF (f IC j exists)
    IF (f IC status = KEPT )
        1/ We calculate local shifts
            bx=\text{cx}(p, jp, 2^{\text{Xmax}}) - \text{cx}(f 2, j - 1, 2^{\text{Xmax}})
            by=\text{cy}(p, jp, 2^{\text{Ymax}}) - \text{cy}(f 2, j - 1, 2^{\text{Ymax}})
            base_x=\text{cx}(p, jp, 2^{\text{Xmax}}) - \text{cx}(f, j, 2^{\text{Xmax}})
            base_y=\text{cy}(p, jp, 2^{\text{Ymax}}) - \text{cy}(f, j, 2^{\text{Ymax}})
        2/ We create a real size temporary fathe f^2 IC_{j-1}
        with null coefficients
        3/ We update it if needed
        IF (f^2 IC_{j-1} exists)
            IF (f^2 IC_{j-1} status = KEPT )
                f^2 IC_{j-1} \_h=0..w-1, b=0..h-1 = f^2 IC_{0,0..w-1,0..h-1}
            ENDIF
        ENDIF
    ENDIF

    4/ We calculate the partial reconstruction
        f^2 IC_{k,1} = f^2 IC_{k,1} + \sum_{n,m} F_F k - 2^n (n + \text{base}_x, l - 2^m (m + \text{base}_y) \_ * f IC_{n,m}
5/ We calculate the sum of all $f^2 tC^{-j-1}$

\[ \text{len} = 2^{X_{\max} + j + 1} \]

\[ f^2 tC^{-j-1} = \text{Local array transfer}\left(p,4^{j_{p}-j+1}, f\ 2 * 4^{j_{p}-j+1}, 1, f^2 tC^{-j-1}, \text{len}\right) \]

6/ If $f^2 lC^{-j-1}$ does not exist we set it to the null vector

7/ We set $f^2 lC^{-j-1}_{\text{status}} = \text{KEPT}$

8/ We transfer the data from $f^2 tC^{-j-1}$ to $f^2 lC^{-j-1}$

\[ f^2 tC^{-j-1}_{0,wh-1,ht-1} \rightarrow f^2 tC^{-j-1}_{hx+0,wh-1,ht+0,ht-1} \]

9/ We don’t need any more $f^2 tC^{-j-1}$ and $f lC^{-j}$

ENDIF

ENDFO

As the function \textit{Local array transfer} is done $jp$ times, we deduce that the transfer complexity of the algorithm is of order $O(2^j p^*(j_{p} + 1))$ and the computational complexity is of order $O(j_{p} \times 2^{X_{\max}} \times 2^{Y_{\max}} / 4^j_{p})$.

If $N=4^{n_{\max}}$ points and $NP=4^{j_{p}}$ processes, the transfer complexity is $O(2^j [\log_4(NP)]^2)$ 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 all the wavelet packets at scale $n_{\max}$, the transfer complexity is $O(2^j p^*(j_{p} + 1))$ and the computational complexity is $O(n_{\max} \times 2^{X_{\max}} \times 2^{Y_{\max}} / 4^j_{p})$.

If we now suppose that $N=4^{n_{\max}}$ points and $NP=4^{j_{p}}$ processes, the transfer complexity is smaller than or equal to $O(2^j [\log_4(NP)]^2)$ and the computational complexity is smaller 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 \( 0 \leq i < 2^{X_{\text{max}}} \) and \( 0 \leq j < 2^{Y_{\text{max}}} \) 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 \( C^j \) wit \( j \in J \) (where \( J \subseteq N_{n_{\text{max}}} = \{ n \in N; 0 \leq n \leq n_{\text{max}} \} \) and \( C^j \subseteq F_j \) (where for each \( j \in J \), \( F_j \subseteq N_{4^{j-1}} \)). In a second part, we decompose the vector \( D \) in all the possible one dimensional wavelet packets \( D^j \) wit \( 0 \leq j \leq n_{\text{max}} \) and \( 0 \leq df_j < 2^j \).

Each frequency \( c^j \) is the combination of a frequency \( c_{x,j} \) among the x-axis and a frequency \( c_{y,j} \) among the y-axis. We write that \( c^j = (c_{x,j}, c_{y,j}) \) and we calculate \( c_{x,j} \) and \( c_{y,j} \) by:

\[
\begin{align*}
    c_{x,j} &= \sum_{i=0}^{j-1} 2^{j-i-1} \left\lfloor \frac{c^j}{4^{j-i-1}} \right\rfloor \mod 2 \\
    c_{y,j} &= \sum_{i=0}^{j-1} 2^{j-i-1} \left\lfloor \frac{c^j}{4^{j-i-1}} \right\rfloor \mod 2
\end{align*}
\]

The multiplication in the wavelet packet expansion consists in obtaining all the packets \( E^j \) solution of the multiplication of \( C^j \) by \( D^j \).

Then, we reconstruct all the packets \( E^j \) to obtain the vector \( E \).
8.5.1 Example of the multiplication in the wavelet packet expansion,
we consider $^{5}E^{3} = (^{3,5}C)^{3} \times ^{3}D^{3}$

**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 beginning, we decompose on $4^{th}$ 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 $^{f}D^{j}$ (named $^{f}lD^{j}$), we obtain the local packet $^{f}lE^{j}$. We insert it in the null packet $^{f}E^{j}$.  

**Figure 8.5.1.1**

![Diagram showing the multiplication process](image_url)
- if \( j \) is greater than or equal to \( j_0 \), we do the multiplication between \( f^j C \) and \( f^j D \) to have \( f^j E \).

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 order 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

- \( 2X_{\text{max}} \) by \( 2Y_{\text{max}} \) is the size of the matrix \( C \)
- \( 2X_{\text{max}} \) is the size of the vector \( D \)
- so \( 2Y_{\text{max}} \) is the size of the vector \( E \)

\( \text{wh}=2X_{\text{max}}-j_0 \) and \( \text{ht}=2Y_{\text{max}}-j_0 \)

**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 \( j_0-1 \)

\( \text{f} = \text{p} / 4^{j+j_0} \)

IF ( \( f^j C \) exists) THEN

IF ( \( f^j C_{\text{status}} = \text{KEPT} \) ) THEN

1/ We set the signal shifts

\[
\text{base}_x = cx(p, j_0, 2X_{\text{max}}) - cx(f, j, 2X_{\text{max}})
\]

\[
\text{base}_y = cy(p, j_0, 2Y_{\text{max}}) - cy(f, j, 2Y_{\text{max}})
\]

2/ We set the frequencies among the \( x \)- and \( y \)-axis
\[
f_x = \sum_{y=0}^{j-1} 2^{-j-y} \left( \left\lfloor \frac{f}{4^{j-y}} \right\rfloor \mod 2 \right)
\]

\[
f_y = \sum_{y=0}^{j-1} 2^{-j-y} \left( \left\lfloor \frac{f}{4^{j-y}} \right\rfloor \mod 2 \right)
\]

3/ We keep the local vector

\[f^x | D^j \mid_{0..wh-1} = f^x | D^j_{\text{base}_{x+0..wh-1}}\]

4/ We perform the smaller matrix-vector multiplication

\[f^x | E^j = f^x | C^j \times f^x | D^j\]

5/ We insert \(f^y | E^j\) into \(f^y | E^j\)

\[f^y | E^j_{\text{base}_{y+0..ht-1}} = f^y | E^j_{0..ht-1}\]

6/ We set \(f^y | E_{\text{status}} = \text{KEPT}\)

ENDIF

ENDIF

ENDFOR j

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

FOR j = jp TO nmax

FOR f = 4jp * p TO 4jp * (p+1) - 1

IF (\(f^j \times C^j\) exists) THEN

IF (\(f^j \times C^j \text{status} = \text{KEPT}\) ) THEN

1/ We perform the smaller matrix-vector multiplication

\[f^y | E^j = f^y | C^j \times f^y | D^j\]

2/ We set \(f^y | E_{\text{status}} = \text{KEPT}\)

ENDIF

ENDIF

ENDFOR f

ENDFOR j

E/ On each process,
We reconstruct from all the kept wavelet packets \( \frac{h_j}{E} \). 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

\[ E = \text{Local array transfer}(p, 4^{0..p}, 0..l, E, 2^{Y_{max}}) \]

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 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 \( O(2^2) \), and if \( R \ll N \) then this method is very efficient.

### 8.6. CONCLUSION