Appendices

During initial part of the research, wavelet transform based techniques were studied by the author. Algorithms developed for wavelet transform computation as part of the research are included as appendices in this thesis. Appendix A describes a fast algorithm based on FFT for discrete wavelet transform and Appendix B details a computational structure and algorithm for wavelet packet decomposition on massively parallel processors machine.
Appendix A

Development of a Modified FFT-Based Algorithm for DWT

A.1 Introduction

The Discrete Wavelet Transform (DWT) [1], in which both time, scale parameters are discrete, has been recognized as a natural wavelet transform for discrete time signals. The demand for real-time operations in many signal processing tasks with large data sets has necessitated fast and computationally efficient algorithms [2, 3, 4, 5, 6] for wavelet transform. Also, many parallel algorithms [7] are available for a variety of parallel processing architectures.

This appendix primarily focuses on the development of an FFT-based algorithm for real-time computation of the DWT. The computational advantage of the proposed algorithm is compared with the FFT-based Fast Wavelet Transform algorithm proposed by Rioul [5] in terms of number of computations per point, for various wavelet kernel size and decomposition levels.

A.2 Computational Structure for Fast Wavelet Transform (FWT)

The computational reorganization proposed by Rioul [5] to reduce the computational load of the well known pyramidal algorithm [8] for DWT and the FFT-based algorithm for its implementation is discussed in this section.

According to the pyramidal structure proposed by Mallat [8], the DWT elementary cell (for each level) contains two filtering operations (a highpass filter $H(z)$ and a lowpass filter $G(z)$),
which are followed by dyadic downsampling. The arithmetic complexity of an FIR filter implementation can be reduced by bringing together the computation of several successive outputs [9]. Since the filter outputs are decimated, the filter bank building blocks can be reorganized [6, 10] based on biphase decomposition (separating into odd and even sequences). The reorganized computational structure is shown in Figure A.1 (a). The z-domain representation of the biphase decomposition of a sequence $y(n)$ is

$$Y(z) = Y_0(z^2) + z^{-1}Y_1(z^2)$$  \hspace{1cm} (A.1)

where $Y_0(z) = \sum_n y[2n]z^{-n}$, the even part and $Y_1(z) = \sum_n y[2n+1]z^{-n}$, the odd part.

Similarly, the biphase decomposition of the $L$-tap filters $G(z)$ and $H(z)$ results in their $L/2$-tap components $G(z) = G_0(z) + G_1(z)$ and $H(z) = H_0(z) + H_1(z)$. Now, the approximation subband for level $j$ can be obtained by

$$Y_g^{j-1}(z) = G_0(z)Y_g^{j-1}(z) + z^{-1}G_1(z)Y_g^{j-1}(z)$$  \hspace{1cm} (A.2a)

and the detail subband

$$Y_k^{j-1}(z) = H_0(z)Y_g^{j-1}(z) + z^{-1}H_1(z)Y_g^{j-1}(z)$$  \hspace{1cm} (A.2b)

where $1 \leq j \leq J$; $J$ is the lowest level of decomposition and $Y_0(z)$ and $Y_1(z)$ are the biphase components of the input sequence $x(n)$. 

Figure A.1 (a) Elementary DWT cell for FWT and (b) FFT-based implementation of DWT cell
A.3 Proposed Modified FFT-Based Algorithm

An FFT-based implementation of the above structure is described next. The input of the DWT cell is given in blocks of \( B \) samples (each \( L/2 \) filter operates on \( B/2 \) length samples) and the wrap-around effect [9] due to cyclic-convolution can be avoided by using overlap-add or overlap-save method. For a filter of length \( L/2 \) and a sequence of length \( B/2 \), the input block-length without wrap-around effect for \( N \) point DFT is,

\[
B = 2N - (L - 2)
\]  

(A.3)

A length-\( N \) FFT of the biphase components of the input sequence and the wavelet scaling filters are computed. Now, four frequency-domain convolutions are performed by multiplying the (Hermitian symmetric) FFT of the input by the (Hermitian symmetric) FFT of the corresponding filter as shown in Figure A.1 (b). The corresponding subsequences are added together for approximation / detail subband and inverse FFT (IFFT) is computed. The approximation subband is used as input for further decomposition. As the data size gets halved at each level due to subsampling, waiting for more blocks from the previous level can be done so that each cell has the same input block length of \( B \) for FFT/IFFT operation. A length-\( N \) FFT is most efficient for an optimized value of the block length.

The signal can be reconstructed from the wavelet representation by transposition of the analysis algorithm and by using synthesis filters \( \hat{H}(z) \) and \( \hat{G}(z) \), which are time-reversed versions of the corresponding analysis filters [5].

A.3 Proposed Modified FFT-Based Algorithm

The proposed algorithm is based on the frequency domain subsampling and makes use of the computational advantage of fast-convolution provided by the structure discussed in Section A.2.

By eliminating the calculation of the approximation subband in levels other than the lowest one of the FWT algorithm, two FFT and one IFFT operations in the intermediate levels can be avoided.

The proposed modified algorithm is explained below. The input for the first level is data samples taken in blocks of length \( B \) (as per Equation (A.3)) and split into even- and odd-
indexed sequences of length $B/2$ (Equation (A.1)). Now, compute $N$ point FFT of these sequences and the initial FFT length $N$ is chosen satisfying the condition, $N \geq 2^{J-1} \cdot L/2$; where $J$ is the maximum decomposition level (division by 2 factor as we use biphase components). As the sequence size reduces by half on entering the next level, the FFT length of the biphase components of the wavelet/scaling filter coefficients are $N/2^{j-1}$ for level $j$. For the first level, the frequency-domain convolutions are performed by multiplying (Hermitian symmetric) FFT of the input by the (Hermitian symmetric) FFT of the wavelet filter and the resulting sequences are added together. Length-$N$ IFFT is applied to obtain the input block’s wavelet coefficients for the first level. For the approximation coefficients, the (Hermitian symmetric) FFTs of the scaling filter are used and the resulting sequences are added together.

Without computing IFFT of the approximation coefficients, decomposition for the lower levels can be done in the Fourier space. If $Y_g^{j-1}[k]$ is a length-$N$ Fourier transform, the length-$N/2$ Fourier transform of its downsampling version is

$$Y_g^j[k] = \sum_{k=1}^{N/2-1} \frac{1}{2} (Y_g^{j-1}[k] + Y_g^{j-1}[k + N/2]) \quad (A.4)$$

The $Y_g^j[k]$ corresponds to the Fourier transform of even samples and that of odd ones have a

---

**Figure A.2 Proposed FFT-based algorithm. (FS stands for Fourier-domain subsampling)**
phase shift [6]. The Fourier-domain approximation coefficients at level \((j-1)\) are downsampled as per Equation (A.4) to halve the resolution and passed as input for next level \(j\). The Hermitian symmetric Fourier-coefficients of the resulting odd / even sequences are multiplied with the corresponding Hermitian symmetric filter \(\text{FFT}_{N/2^{j-1}}\) to get the detail / approximation coefficients for level \(2 \leq j \leq J\). The length-\(N/2^{j-1}\) IFFT can be done in each level \(j\) for the wavelet coefficients to get the detail subband and in the last level, length-\(N/2^{J-1}\) IFFT is used for approximation subband computation. The coefficients from various blocks are grouped together to get the wavelet / approximation subband. So, every level, except the first and last, has only one IFFT operation (for detail subband) other than complex multiplication and block addition operations. A two level decomposition implementation of the proposed algorithm is shown in Figure (A.2).

Since each block at any stage is processed independently, the algorithm has no inter-block dependency as in the case of FWT. So, no hidden synchronization overhead is involved in the proposed algorithm implementation.

For the synthesis part, the transposed flow graph of the analysis algorithm shown in Figure (A.2) can be used with synthesis filter components which are time-reversed versions of analysis filters.

\section*{A.4 Computational Complexity}

The computational complexity in terms of the number of real multiplications and real additions required by the candidate algorithms is calculated in this section. The total number of operations (multiplications + additions) is considered as the appropriate criterion for performance comparison of various algorithms [5].

The “split-radix” FFT [11] used in both algorithms has best known complexity for length \(N = 2^k\). For real data, the split-radix FFT (or inverse FFT) requires exactly

\[2^{k-1}(k - 3) + 2 \text{ (real) multiplications}\]

and

\[2^{k-1}(3k - 5) + 4 \text{ (real) additions}\]
The filters \((H(z) \text{ and } G(z))\) in the computation of the DWT usually have equal length. The filter FFT's can be pre-computed and applied as and when needed.

The FWT algorithm has a regular computational structure. The operations required by an elementary cell are counted as follows [5]. The four frequency-domain convolution operations require \(4N/2\) complex multiplications. Assuming that a complex multiplication is done with three real multiplications and three real additions [9], the computational complexity of an elementary cell can be expressed as

\[
2 \text{ FFT}_N + 4 \cdot 3 \cdot N/2 \text{ mults} + 4 \cdot 3 \cdot N/2 \text{ adds} + 2 \cdot N/2 \text{ adds} + 2 \text{ IFFT}_N
\]

This can be simplified as \(k2^{k+1} + 8\) multiplications and \((3k-1)2^{k+1} + 16\) additions [5]. The total number of elementary cells required for depth \(J\) decomposition is \(2 \cdot (1 - 2^{-J})\). So, the total number of multiplications required per point is

\[
M_1 = \frac{((k2^{k+1} + 8) \cdot 2 \cdot (1 - 2^{-J}))/B}{(A.7)}
\]

and the total number of additions per point is

\[
A_1 = \frac{((3k-1)2^{k+1} + 16) \cdot 2 \cdot (1 - 2^{-J}))/B}{(A.8)}
\]

In the case of proposed algorithm, the first level has two length-\(N\) FFT and one length-\(N\) IFFT. The last level has two length-\(N/2^{J-1}\) IFFT and all the intermediate levels has one length-\(N/2^{J-1}\) IFFT computations per input block. The frequency-domain convolution complexity is \(4 \cdot (N/2^j)\) and block addition complexity is \(2 \cdot (N/2^j)\), where \(1 \leq j \leq J\). The frequency-domain downsampling operation for level \(j\) requires \(N/2^j\) additions. The total operations on an input block for decomposition depth \(J\) can be expressed as

\[
(2 \text{ FFT}_N)_1 + (2 \text{ IFFT}_N\cdot 2^{J-1})' + (\text{IFFT}_{N \cdot 2^{J-1}})_{1 \rightarrow (J-1)} + \left(\frac{N}{2^j}\right)_{1 \rightarrow (J-1)} \text{ adds} + (4 \cdot 3 \cdot \frac{N}{2^j})_{1 \rightarrow J} \text{ mults}
\]

\[
+ (4 \cdot 3 \cdot \frac{N}{2^j})_{1 \rightarrow J} \text{ adds} + (2 \cdot \frac{N}{2^j})_{1 \rightarrow J} \text{ adds} + (2 \cdot \frac{N}{2^j})_{1 \rightarrow (J-1)} \text{ adds}.
\]
The subscript of parenthesis denotes the level in which the operations are performed. The total number of multiplications per point, using Equation (A.5), is

\[ M_2 = \left( (2^k(k-3)+4) + (2^{k-j}(k-J-2)+2 + \sum_{j=1}^{j} 2^{k-j}(k-j+10)+2) \right) / B \quad (A.9) \]

The total number of additions per point, using Equation (A.6), is

\[ A_2 = \left( (2^k(3k-5)+8) + (2^{k-j}(3(k-J)-2)+4) + (\sum_{j=1}^{j} 3*2^{k-j}(k-j+4)+4) + (\sum_{j=1}^{j} 2^{n-j}) \right) / B \quad (A.10) \]

The computational complexity per point of the FWT and that of the proposed algorithm are calculated using the above equations. Appropriate initial FFT length which gives best performance for a given algorithm is chosen for a particular wavelet kernel size. The results are detailed below.

### A.5 Results and Discussions

Table A.1 lists the resulting number of real multiplications per input point required by the candidate algorithms for various wavelet kernel sizes at different decomposition depths. The proposed algorithm has less number of multiplications per point for filter size greater than four and decomposition depth greater than one. Also, the performance improves with an increase in decomposition depth.

Table A.2 lists the number of real additions per point required for both algorithms. The number of real additions is less for the proposed algorithm compared to FWT for filter size greater than two. The same trend as in Table A.1 can be seen regarding the improvement in addition complexity also with an increase in wavelet kernel size and level.

Although both Vetterli's algorithm [6] and the proposed algorithm uses Fourier-domain subsampling, the latter has better performance due to the use of subsampled sequences for initial FFT computation (FFT length being more close to the best performance length) and Hermitian symmetry property.
Appendix A. Development of a Modified FFT-Based Algorithm for DWT

### Table A.1 FFT-Based DWT algorithms: multiplication complexity per point*

<table>
<thead>
<tr>
<th>Filter Length</th>
<th>LEVEL 1</th>
<th>LEVEL 2</th>
<th>LEVEL 3</th>
<th>LEVEL 4</th>
<th>LEVEL 5</th>
<th>FFT LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>II</td>
<td>I</td>
<td>II</td>
<td>I</td>
<td>II</td>
</tr>
<tr>
<td>2</td>
<td>3.0</td>
<td>5.1</td>
<td>4.5</td>
<td>6.4</td>
<td>5.2</td>
<td>7.0</td>
</tr>
<tr>
<td>4</td>
<td>4.0</td>
<td>5.2</td>
<td>6.0</td>
<td>6.6</td>
<td>7.0</td>
<td>7.2</td>
</tr>
<tr>
<td>8</td>
<td>5.2</td>
<td>6.3</td>
<td>7.8</td>
<td>7.6</td>
<td>9.1</td>
<td>8.3</td>
</tr>
<tr>
<td>16</td>
<td>6.5</td>
<td>7.4</td>
<td>9.8</td>
<td>8.7</td>
<td>11.8</td>
<td>9.4</td>
</tr>
<tr>
<td>32</td>
<td>7.9</td>
<td>8.5</td>
<td>11.8</td>
<td>9.8</td>
<td>13.8</td>
<td>10.5</td>
</tr>
<tr>
<td>64</td>
<td>9.1</td>
<td>9.5</td>
<td>13.6</td>
<td>10.9</td>
<td>15.9</td>
<td>11.5</td>
</tr>
<tr>
<td>128</td>
<td>10.2</td>
<td>10.6</td>
<td>15.4</td>
<td>12.0</td>
<td>17.9</td>
<td>12.6</td>
</tr>
</tbody>
</table>

*Each entry gives the number of real multiplications per input point for various decomposition levels. The notations I and II represent the FWT algorithm and the proposed algorithm respectively. The last column shows the corresponding initial FFT length.

### Table A.2 FFT-Based DWT algorithms: addition complexity per point*

<table>
<thead>
<tr>
<th>Filter Length</th>
<th>LEVEL 1</th>
<th>LEVEL 2</th>
<th>LEVEL 3</th>
<th>LEVEL 4</th>
<th>LEVEL 5</th>
<th>FFT LENGTH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>I</td>
<td>II</td>
<td>I</td>
<td>II</td>
<td>I</td>
<td>II</td>
</tr>
<tr>
<td>2</td>
<td>6.0</td>
<td>13.7</td>
<td>9.0</td>
<td>15.0</td>
<td>10.5</td>
<td>15.7</td>
</tr>
<tr>
<td>4</td>
<td>9.3</td>
<td>14.1</td>
<td>14.0</td>
<td>15.5</td>
<td>16.3</td>
<td>16.2</td>
</tr>
<tr>
<td>8</td>
<td>14.1</td>
<td>17.4</td>
<td>21.2</td>
<td>18.7</td>
<td>24.7</td>
<td>19.4</td>
</tr>
<tr>
<td>16</td>
<td>18.2</td>
<td>20.6</td>
<td>27.3</td>
<td>22.0</td>
<td>31.9</td>
<td>22.7</td>
</tr>
<tr>
<td>32</td>
<td>22.3</td>
<td>23.9</td>
<td>33.5</td>
<td>25.2</td>
<td>39.1</td>
<td>25.9</td>
</tr>
<tr>
<td>64</td>
<td>26.2</td>
<td>27.1</td>
<td>39.3</td>
<td>28.4</td>
<td>45.8</td>
<td>29.1</td>
</tr>
<tr>
<td>128</td>
<td>29.6</td>
<td>30.4</td>
<td>44.5</td>
<td>31.7</td>
<td>51.9</td>
<td>32.4</td>
</tr>
</tbody>
</table>

*Each entry gives the number of real additions per input point for various decomposition levels. The notations I and II represent the FWT algorithm and the proposed algorithm respectively. The last column shows the corresponding initial FFT length.

### A.6 Conclusion

A computationally efficient FFT-based DWT algorithm is presented in this appendix. The FWT algorithm [4] has been proved to be better in performance than pyramidal algorithm by Mallat [8] and FFT-based Vetterli's Algorithm [6]. The computational complexity calculations show that the proposed algorithm provides remarkable savings for wavelet kernel size greater than four (which are widely used), compared to FWT. Also, the performance of the algorithm increases with decomposition depth. The lack of inter-block dependency is a useful feature in parallel processing environment. The proposed algorithm is best suited for computationally intensive applications, such as in image processing.
References:


Appendix B

Development of a Computational Structure for Fast Computation of Wavelet Packet Transform on MPPs

B.1 Introduction

In this appendix, a Parallel Multiple Subsequence (PMS) structure is developed for wavelet packet (WP) decomposition. In PMS structure, subbands are computed using subsequences obtained directly from the input data, improving parallelism in computation. An algorithm for implementation of PMS on massively parallel processors (MPPs) is also developed.

Wavelet packets, which comprise of the entire family of subband coded decompositions, is an ideal tool in multiresolution analysis. In the wavelet transform [1] computation, the signal is decomposed into coarse scale approximations and the detail signal. This procedure is applied recursively to the coarse scale approximations leading to the well known filter bank tree wavelet decomposition structure. In the WP decomposition the recursive procedure is applied to both coarse scale approximation and detail signals, which leads to a complete binary tree, giving more flexibility in frequency resolution.

Several efficient parallel algorithms [2, 3] proposed for the fast wavelet transform are applicable to WP decomposition also. Some of the works in parallel wavelet packet decomposition includes subband based approaches for performing the best basis selection on

But, most of these algorithms are based on the filter bank tree structure. The delay associated with the implementation grows exponentially with the number of levels [9]. For instance, the set of basis functions for Short Time Fourier representation of a signal requires the lowest level WP subbands only. With filter bank tree structure, one has to perform unnecessary computations by way of evaluating the higher level subbands. One of the important factors limiting the range of scalability in parallel processing is the sequential component of the algorithm [2].

**B.2 Wavelet Packet Transform Algorithms**

This section briefly describes the filter bank tree algorithm by Mallat [10] and proposed PMS structure [11] based algorithm for WP decomposition is then explained.

**B.2.1 The Filter Bank Tree WP Algorithm**

The wavelet packet decomposition extends the discrete wavelet transform in a way that each level $j$ consists of $2^j$ subbands, generated by a tree of low pass and high pass operations. Consider the analysis filter bank of the 1-D WP scheme shown in Figure B.1. In this figure, the analysis filters $H(z)$ represents a high pass filter and $G(z)$ represents a low pass filter. The WP transform of a discrete signal $x(n)$ can be computed by convolving with filters $H(z)$ and $G(z)$ followed by dyadic downsampling. This process is repeated on both sequences until the required

![Figure B.1 Two level WP decomposition using filter bank tree algorithm](image)

Figure B.1 Two level WP decomposition using filter bank tree algorithm
B.2 Wavelet Packet Transform Algorithms

level of decomposition is reached.

The WP subbands at any level $j$ are given by

$$X_{2j-1}^i(n) = \sum_k h(k) X_{2j-1}^{i-1}(2n - k) \quad \text{(B.1a)}$$

$$X_{2j}^i(n) = \sum_k g(k) X_{2j}^{i-1}(2n - k) \quad \text{(B.1b)}$$

where $X_0^n = x(n)$; the input sequence $(n \in \mathbb{Z})$, $j = 1, 2, \ldots, J$; denotes different levels and $1 \leq i \leq 2^{j-1}$ is the subband index within a level.

B.2.2 Parallel Multiple Subsequence (PMS) Structure Based Algorithm

The PMS structure [11], originally developed for DWT, is based on the principle of polyphase splitting for subband decomposition. Here, an extension to the PMS structure for WP transform is developed. From the wavelet (defined by its filter $H(z)$) and its smoothing function (defined by its filter $G(z)$) we compute the filter coefficients for the subbands at each level by successive convolutions and upsampling. Subbands at various levels are computed directly by convolving the corresponding filter with the original data.

The subbands can be computed based on the PMS structure as follows

$$X_{2j-1}^i(k) = \sum_{p=1}^{2^i} x_{j,p}(k) * h_{i,p}(-k) \quad \text{(B.2a)}$$

$$X_{2j}^i(k) = \sum_{p=1}^{2^i} x_{j,p}(k) * g_{i,p}(-k) \quad \text{(B.2b)}$$

where $\ast$ denote convolution and $1 \leq i \leq 2^{j-1}$ is the subband index,

$x_{j,p}(k) = x(2^j k + p - 1)$,

$h_{i,p}(k) = h_i^j(2^j k + 2^j - p + 1)$, and

$g_{i,p}(k) = g_i^j(2^j k + 2^j - p + 1)$.

The PMS structure for second level WP decomposition is shown in Figure B.2 Being a regular structure, this can be extended to any level. The PMS structure has got parallelism both within and between levels, making it highly suitable in parallel processing environment.
Appendix B. Development of a Computational Structure for Fast Computation...

Figure B.2 Second level WP decomposition using PMS structure

B.3 Algorithm Analysis

The scalability and computational complexity of the algorithms described in Section (B.2) are analyzed on coarse-grained machines. The platform used is a distributed memory...
B.3 Algorithm Analysis

architecture in which each processor has fast access to local memory

B.3.1 Computational Model and Assumptions

The notion of scalability of an algorithm and parameters of the computational model are defined based on references [12, 13]. Let $t_f$ be the time required for one floating point operation. The time required for the complete transfer of a message containing $m$ words between two processors that are $l$ connections away is given by the $t_s + (t_w m) * l$, where $t_s$ is the startup time, and $t_w = \text{bytes-per-word} / B$, where $B$ is the bandwidth of the communication channel between the processors in bytes per second. So, the total execution time mainly consists of two parts: one corresponding to the computation complexity and the other corresponding to the communication complexity.

Let $T(n,p)$ be the time taken by an algorithm on a $p$ processor architecture with input data size $n$. The algorithm is considered scalable on the structure if $T(n,p)$ increases linearly with an increase in the data size or decrease linearly with the increasing number of processors (machine size). We assume that $p < n$, as we are interested in large problem sizes generally.

The performance study of the algorithms is done by varying the machine size ($p$), problem size ($n$) and the wavelet filter kernel size ($L$) for different levels of decomposition. For the sake of simplicity in analysis, we assume that the problem size and machine size are powers of 2, i.e., $2^n$ and $2^a$ respectively. The scalability and performance in parallel environment is analyzed for generating subbands at a given level only.

B.3.2 Data Distribution Strategy

The main problem faced, when dealing with multicomputers, is how to perform an efficient mapping of tasks and data to the processors which raises the questions of load balancing and communication minimization. Both questions are closely connected with the data distribution task. For both the algorithms, two methods of handling border data ($L$ coefficients of the neighboring Processing Elements (PE) are required to compute a single output coefficient at the border) can be used [14]. They are

- **Data Swapping**: Each PE computes only non-redundant data and then exchanges these results with the appropriate neighboring PEs, in order to get the necessary data for the next
calculation step (i.e., the next decomposition level).

- **Data Overlapping:** In the initialization step, each PE is provided not only its share of the original signal but also the data set which is required to compute the redundant data. This avoids additional communication with neighbor PEs to obtain the border data. Appropriate data distribution scheme is chosen in the analysis for a given algorithm.

## B.3.3 Analysis of the Filter Bank Tree Algorithm

The parallel implementation algorithm used here is based on WP image decomposition algorithm proposed by Feil and Uhl [5]. It is found that for data distribution, in a filter bank tree algorithm, the data overlapping approach is not competitive at all over a wide range of different architectures [6]. So, the data distribution scheme used here will be Data swapping method.

The most natural way to distribute the computational work of a WP transform can be found on a distributed memory architecture with the number of PEs equal to a power of 2, i.e., \( p = 2^a \). The input data \( 2^n \) for each level (will be approximately the same as the original input data, ignoring the increase in length caused by convolution, as all subbands are retained in WP decomposition) is partitioned into \( 2^a \) parts of equal size \( 2^{a-a} \). The partitioning is done in two different ways depending upon whether level \( j \) is smaller or larger than \( a \). Let \( i \) denote the subband index and \( 0 \leq i < 2^j \). If \( j < a \), a subband with index \( i \) is not assigned to a single PE but is shared by PEs with processor index in the range \( 2^{a-j} \times i \) to \( 2^{a-j} \times (i + 1) - 1 \). Therefore, in the initialization step, those \( 2^{a-j} \) PEs will exchange their data in order to have the entire shared subband residing on each of them. Then, in the second step, they will calculate their own part of the subband they share at level \( j + 1 \). If \( j \geq a \), \( 2^{j-a} \) subbands and also their two children reside on each PE. Thus no communication among PEs is needed for the subset of subbands residing on each PE at level \( j \).

The message communication required for level \( j \) is \( L \) (filter length) data units across \( 2^{a-j+1} - 1 \) PEs for subband computation and \( 2^{a-a} \) data units across \( 2^{a-j} \) PEs for data redistribution on entering a new level. This is required for all the \( 2^j \) subbands of the level. Thus, the overall communication amount (i.e., the number of datapoints sent) can be expressed as
The total number of PEs involved in the message transfer at various stages is

\[ k = \sum_{j=1}^{a-1} 3 * 2^a - 2^j \].

Based on the parameters described in Section B.3.1, the total time required for message communication is \( k * t_s + m * t_w \). The computation of each output coefficient requires \( 2L \) floating point operations (additions and multiplications). As each processor holds \( 2^{n-a} \) data units, the total computation time is \( 2L * 2^{n-a} * J * t_s \), where \( J \) is the maximum decomposition level. Thus the total time taken for WP decomposition is given by,

\[ T_i = (2L * 2^{n-a} * J * t_f) + (k * t_s + m * t_w) \]  

(B.4)

**B.3.4 Analysis of the PMS Structure Based Algorithm**

As the PMS structure is tailored for the parallel computation of the subbands of a given level directly from the original input sequence, there is no sequential part in the algorithm. So, the data distribution scheme proposed for PMS structure based algorithm is Data Overlapping approach i.e., all necessary data desired to compute the subbands is sent to the processors initially. The proposed data distribution strategy is outlined below.

The number of subbands in a regular WP decomposition scheme is \( 2^j \) for level \( j \). But, PMS structure splits each subband (and the corresponding filter) again into \( 2^j \) subsequences. This results in \( 2^j * 2^j \) subsequences for the level \( j \). The input data is partitioned into \( 2^a \) parts of equal size \( 2^{n-a} \). The data partitioning can be done in two different ways depending upon whether \( 2^j \) is smaller or larger than \( a \). Let \( i \) denote the subsequence index and \( 0 \leq i < 2^2^j \). If \( 2^j < a \), the number of subsequences is less than the number of available PEs and each subsequence with an index \( i \) is not assigned to a single PE but is shared by PEs with processor index in the range \( 2^{a-2j} * i \) to \( 2^{a-2j} * (i + 1) - 1 \). The redundant data units to be distributed initially among PEs is \( L_j / 2^j \), where \( L_j = (L - 1)(2^j - 1) + 1 \); the filter length for level \( j \). As each PE is having the entire data units required, no message communication need to be performed in this distribution scheme and the computational work is uniformly distributed. If \( 2^j \geq a \),
2^{j-a} subsequences can reside on each PE. Then, initial redundant data distribution is also not required.

The computation of each output coefficient requires \(2 \cdot (L_j / 2^j) \cdot 2^j = 2L_j\) floating point operations. Since each processor holds \(2^{n-a}\) data units, the total computation time is \(2L_j \cdot 2^{n-a} \cdot t_f\). As there is no message passing required, the total time taken for WP decomposition is

\[
T_2 = 2L_j \cdot 2^{n-a} \cdot t_f
\]  

(B.5)

### B.4 Analytical Results and Discussion

In order to get an approximate figure of the timings, the system parameters of Intel Paragon XP / S machine [2] is used in equation (B.4) and (B.5). The paragon machine has a 2-D mesh (torus) connection structure with support for number of processors in the range of 64 - 4000. The per node memory capacity is 128 MB. The communication bandwidth of the machine is 200 MB/s. Each processor has a peak performance (64 bits) of 75 Mflop / s and the communication latency is around 100 \(\mu\)secs. The performance measurement criterion used here is speedup, which is taken as the ratio of execution time of the filter bank tree algorithm to that of the PMS structure based algorithm, i.e.,

\[
\text{speedup} = T_1 / T_2
\]  

(B6)

Figure B.3 compares the scalability of the candidate algorithms for increasing machine and problem size. Figure B.3(a) plots the execution time of filter bank tree and PMS based algorithms at a decomposition depth of 6 on various machine size with fixed wavelet kernel size \((L = 16)\) for a problem size of 128 MB. The execution time decreases for the PMS based algorithm with an increase in machine size whereas it linearly increases for the filter bank tree algorithm due to the communication overhead. Figure B.3(b) shows the execution time for various problem sizes on 512 processors of the Paragon using a 16-tap wavelet kernel for various problem sizes. It can be noted that the execution time increases linearly with the problem size and hence PMS algorithm perfectly satisfies the scalability criterion. Although the execution time for both the algorithms increases with problem size, due to the communication
B.4 Analytical Results and Discussion

Figure B.3 Comparison of scalability of filter bank tree and PMS algorithm at decomposition depth 6 and filter kernel size 16 for (a) different machine size and (b) different problem size.

Figure B.4 Comparison of performance (speedup) of filter bank tree and PMS algorithm at different decomposition depth and fixed problem size of 128 MB. (a) filter kernel size 16 on different machine size (b) number of PEs 1024 and different filter kernel size.

overhead caused by the data re-distribution between the levels, the rate of increase of filter bank tree algorithm is much faster than that of PMS.

Figure B.4 shows the speedup of the proposed algorithm over the filter bank tree algorithm for various decomposition levels. The speedup increases significantly with machine size up to level 8 as shown in Figure B.4 (a). The plot of speedup for a usual range of filter length, 32, at a
machine size of 512 is given in Figure B.4 (b). This figure also indicates that even at a decomposition depth of 10 the speedup value is 2, which is very promising.

The timing calculations did not take into account the practical runtime delay factors such as network congestion. But, excluding these factors favors only filter bank tree algorithm as no inter-processor message transfer is demanded by the PMS based algorithm. The results obtained suggest that the proposed algorithm is superior to filter bank tree algorithm on massively parallel processors for lowest level wavelet packet subband decomposition. Besides, the proposed algorithm has much better performance for large problem sizes.

### B.5 Conclusion

An efficient and scalable computational structure and its parallel implementation for WP decomposition on massively parallel processors with distributed memory were developed. The analytical study shows considerable speedup of the PMS structure based algorithm in comparison with filter bank tree based algorithm. As no inter-processor communication overhead is involved in PMS based algorithm, it provides architectural and algorithmic scalability. Due to increase in communication overhead with the machine size and problem size, the filter bank tree algorithm is not perfectly scalable. The PMS structure based algorithm is useful for applications like in numerical mathematics and Short Time Fourier Transform basis representation.

### References:


