Wavelet Image and Video Coding on Parallel Architectures

M. Feil, R. Kutil, P. Meerwald, and A. Uhl
RIST++ and Dept. of Scientific Computing
University of Salzburg, Austria

Abstract

We discuss parallel algorithms for wavelet-based image and video coding. After reviewing fundamentals of the parallel discrete wavelet transform we cover the parallelization of two state-of-the-art compression schemes: a C++ 3-D SPIHT video codec and a JAVA JPEG-2000 implementation.

1 Introduction

In recent years there has been a tremendous increase in the demand for digital imagery. Applications include consumer electronics (Kodak's Photo-CD, HDTV, SHDTV, Video-on-Demand, and Sega's CD-ROM video game), medical imaging (digital radiography), video-conferencing and scientific visualization. The problem inherent to any digital image or digital video system is the large bandwidth required for transmission or storage.

Unfortunately, many compression techniques and applications demand execution times that are not possible using a single serial microprocessor [85], which leads to the use of high performance computers for such tasks (beside the use of DSP chips, FPGAs, or application specific VLSI designs). In this context, several papers have been published describing real-time image and video coding on general purpose parallel architectures – see for example JPEG [6, 15, 23], MPEG-1,2,4 [1, 2, 43, 86], H.261 [21, 59, 103], H.263 [3], vector quantization [63, 65], and fractal compression [42, 46, 47, 79, 97].

Image and video coding methods that use wavelet transforms have been successful in providing high rates of compression while maintaining good image quality and have generated much interest in the scientific community as competitors to DCT based compression schemes. With the finalization of the wavelet based JPEG2000 standard [9, 11] and the inclusion of a wavelet algorithm for synthetic/natural hybrid coding in MPEG-4 [87] there is no doubt left that wavelet compression has to be considered state of the art nowadays. Therefore, a thorough investigation of parallel versions of these algorithms seems mandatory.

As a first step for an efficient parallel wavelet image and video compression algorithm, the wavelet decomposition has to be carried out (followed by subsequent quantization and entropy coding of the transform coefficients). With respect to computational demand, the transform step is a major part of each wavelet-based compression algorithm. Therefore, the search for efficient parallel wavelet transform algorithms is mandatory to guarantee good overall performance of a corresponding parallel image or video codec.

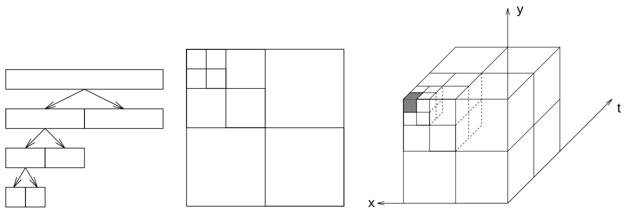
In Section 2, we shortly review the algorithmic principles of the fast wavelet transform and some major wavelet-based image and video compression techniques. Section 3 discusses parallel algorithms for the fast wavelet transform and wavelet packet decompositions on MIMD architectures. Section 4 covers specific parallel image and video codecs – in particular, we focus on a message passing based MIMD parallelization of a 3-D SPIHT video codec and a thread-based JAVA parallelization of JPEG2000.

2 Wavelet-based Image and Video Compression

2.1 Fast Wavelet Transform and Wavelet Packet Decomposition

The fast wavelet transform can be efficiently implemented by a pair of appropriately designed Quadrature Mirror Filters (QMF). A 1-D wavelet transform of a signal S is performed by convolving S with both QMF's and downsampling by 2; since S is finite, one must make some choice about what values to pad the extensions with. This operation decomposes the original signal into two frequency-bands (called subbands), which are often denoted coarse scale approximation and detail signal. Then the same procedure is applied recursively to the coarse scale approximations several times (see Fig. 1.a).

The classical 2-D transform is performed by two separate 1-D transforms along the rows and the columns of the image data S, resulting at each decomposition step in a low pass image (the coarse scale approximation) and three detail images (see Fig. 1.b). To be more concise, this is achieved by first convolving the rows of the low pass image S_{j+1} (or the original image in the first decomposition level) with the QMF filterpair G and H (which are a high pass and a low pass filter, respectively), retaining every other row, then convolving the columns of the resulting images with the same filterpair and retaining every other column. The same procedure is applied again to the coarse scale approximation S_j and to all subsequent approximations.



(a) 1D (b) 2D (c) 3D

Figure 1. Pyramidal wavelet decomposition

By analogy to the 2-D case the 3-D wavelet decomposition is computed by applying three separate 1-D transforms along the coordinate axes of the 3-D data. As it is the case for 2-D decompositions, it does not matter in which order the filtering is performed (e.g. a 2-D filtering frame by frame with subsequent temporal filtering, three 1-D filterings along y, t, and x axes, e.t.c.). After one decomposition step we result in 8 frequency subbands out of which only the approximation data (the gray cube in Fig. 1.c) is processed further in the next decomposition step. This means that the data on which computations are performed are reduced to $\frac{1}{8}$ in each decomposition step.

Wavelet packets [99] represent a generalization of the method of multiresolution decomposition and comprise the entire family of subband coded (tree) decompositions. Whereas in the wavelet case the decomposition is applied recursively to the coarse scale approximations, in the wavelet packet decomposition the recursive procedure is applied to all the coarse scale approximations and detail signals, which leads to a complete wavelet packet tree (i.e. binary tree and quadtree in the 1-D and 2-D case, respectively) and more flexibility in frequency resolution. See Fig. 2 for the 2-D case.

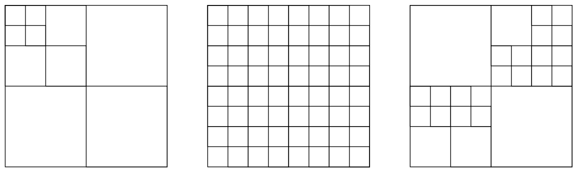


Figure 2. Pyramidal wavelet decomposition, wavelet packet decomposition with uniform time-frequency resolution, and wavelet packet decomposition with arbitrary subband structure

2.2 Wavelet-based Image and Video Codecs

Image compression methods that use wavelet transforms [92] (which are based on multiresolution analysis – MRA) have been proven to deliver the best rate-distortion performance for texture compression of all currently available codecs. A wide variety of techniques to process the wavelet transform domain further for compression purposes have been reported in the literature [5, 61], ranging from simple entropy coding to more complex techniques such as vector quantization [4, 17], adaptive transforms [20, 95], zero-tree encoding [84], and edge-based coding [36].

The most advanced and well-known compression algorithms in the wavelet area are SPIHT [82] and EBCOT [90] (which is the basis of JPEG2000).

The wavelet packet "best basis algorithm" [14] performs an adaptive optimization of the frequency resolution of a complete WP decomposition tree by selecting the most suitable frequency subbands for signal compression. This is done by optimizing additive information cost functions. The same algorithm employed with non-additive cost function is denoted "near-best basis algorithm" [89], if the subband structure is restricted to uniform time-frequency resolution the corresponding algorithm is denoted "best level selection".

In the context of image compression a more advanced technique is to use a framework that includes both rate and distortion, where the best basis subtree which minimizes the global distortion for a given coding budget is searched [80]. Other methods use fixed bases of subbands for similar signals (e.g. fingerprints [44]) or search for good representations with a genetic algorithm [7, 12, 83]. Recently, wavelet packet based compression methods have been developed [66, 68, 101] which outperform the most advanced wavelet coders significantly for certain image types in terms of rate-distortion performance.

A significant amount of work has also been devoted to wavelet/subband based video coding (see e.g. [38, 49, 60, 91] for 3-D wavelet/subband coding and [10, 50, 67, 106, 105] and Chapter 20, 21 of [92] for 2-D coding with motion estimation).

3 Parallel Fast Wavelet Transform: A Review

A vast amount of work has been devoted to implement and analyze parallel algorithms for wavelet transforms. However, since not even the term "wavelet" is a very specific one it is obvious that different kinds of wavelet transforms exist. Depending on the type of target application one may choose among a multitude of different wavelet transforms, each of them associated with certain advantages and disadvantages. For example, only the "fast wavelet transform" (also denoted discrete wavelet transform (DWT) or pyramidal wavelet decomposition) is well suited for real-time applications and all kinds of compression purposes. Therefore, it is hard to judge the content of a paper devoted to "parallel wavelet transforms". There have been papers carrying that title devoted to parallel Gabor transform [64, 69, 70], continuous wavelet transform with arbitrary time-frequency resolution [32, 33, 37, 94], the à trous algorithm [8, 24, 26, 27, 30, 58, 78], non-standard wavelet decompositions for numerical applications [35], and of course to the classical fast wavelet transform or DWT.

Here we focus onto algorithmic questions specifically related to MIMD DWT wavelet transform algorithms and provide pointers to the literature devoted to this topic. Usually, the data are distributed in some way among the processors available and the DWT is applied onto local data. In this setting, the two main issues with respect to parallel DWT are as follows:

- Handling of border data
- Data decomposition strategies

When distributing the data to be transformed among several processor elements (PEs), handling of border data needs to be specified in some way since the wavelet filters overlap the borders between adjacent data parts when performing the filtering across the data boundary. In order to provide the necessary border data to each PE after an initial data distribution we may distinguish between two approaches for border treatment (compared theoretically and experimentally for 1-D and 2-D DWT in [24, 35, 88, 100] and in [55] for 3-D DWT) which trade off communication against computational complexity:

- *Data swapping* method (also known as *non-redundant data calculation*): each PE computes only on local data and exchanges these results with the appropriate neighbour PE in order to get the necessary data for the next calculation step (i.e. the next decomposition level).
- *Redundant data calculation* approach: in the initialization step we do not only provide to a PE its share of the original signal but broadcast the entire data set to all PE (which limits this approach to moderately sized data sets). Subsequently, in each calculation step each PE computes also redundant data in order to avoid additional communication with neighbour PEs to obtain the required border data.

An algorithm without taking care of border data at all is presented in [102]. Although it is claimed that this has minor influence on applications, this approach is not recommended for compression purposes. In most papers, the data swapping approach is preferred (e.g., [22, 54]). [55, 100] investigate the optimal decomposition level to switch from redundant data calculations to data swapping, provide computation and communication cost estimates of both techniques, and give experimental results on SGI Origin, Power Challenge, and Intel Paragon, respectively.

A third approach used in higher dimensions to provide the border data is to perform all decomposition steps corresponding to one coordinate direction, than to do a global data transposition, and finally compute the remaining directions. This approach is compared to data swapping in [52, 76].

Now we proceed with the discussion of data decomposition strategies. Whereas there is nothing to discuss about data decomposition in the 1-D case, different possibilities exist for the 2-D and 3-D cases. The main distinction is among stripe partitioning and checkerboard partitioning (which are the 2-D cases, in the 3-D case simply a third dimension is added). Whereas checkerboard partitioning offers the obvious advantages of minimizing the block-border length at the cost of a larger number of neighbouring blocks, stripe partitioning requires only communication with two direct neighbours. Papers providing comparisons of these techniques are [35, 75, 77, 102, 55]. In more detail, [35] gives an in-depth scalability analysis of both approaches for the 2-D case based on a previous data dependence analysis [34]. [55, 75, 102] treat the 3-D case and present experimental results for Cray T3D, SGI workstations and SGI Power Challange. Most papers favor and use the striping approach, e.g.

- [22] suggests a striping adapted to a snake pattern processor arrangement and gives results on Cray T3D and Intel Paragon
- [52] gives time complexity estimates and compares the results to the abovementioned transpose approach, experiments are conducted on CM-5
- [77] provides complexity estimates for both stripe and checkerboard partition and gives experimental results on Intel Paragon

A checkerboard based image partitioning scheme employing PVM and the wavelet lifting scheme is presented in [98].

Additionally, several more specialized topics have been covered in the literature. [104] describes an adaptive data distribution following the processor speed in a heterogeneous network using PVM. Optimization of 1-D DWT for multicomputers with De Bruijn Graph network topology is given in [19]. An algorithm to compute redundant data (i.e. each of four PEs carries an entirely different subband) suited for hypercubes (nCube) is derived in [45]. [51] gives a 1-D CREW PRAM complexity estimation, [57] investigates a data arrangement strategy using a space filling scan for optimizing cache and memory use to match the filter process, and [48] presents a 2-D DWT on a multithreaded architecture and gives a comparison to a MPI implementation. Cache problems of 3-D DWT are discussed and solved in [54].

Compared to the fast wavelet transform, only relatively few papers have been devoted to parallel wavelet packet decomposition and its specific features and demands. In this context it has turned out [25, 39, 56, 93, 96] that it is advisable to to perform a subband based data decomposition instead of the concepts mentioned before at a certain stage of the computation. This is explained briefly for the 2-D case. Fig.3 shows the data arrangement for levels $j = 0 \ldots 2$: at level j we get 4^j subbands, each with 2^{Xmax-j} by 2^{Ymax-j} coefficients.

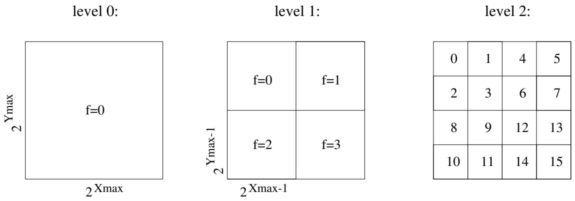


Figure 3. Data arrangement and labeling for a 2-D wavelet packet decomposition

To do the decomposition in parallel, the data is redistributed according to the subband structure (after an initial stripe or checkerboard distribution - see Fig. 4 on the left side) at that specific decomposition level (denoted “distribution level” [25, 96]) where the number of PEs is lower or equal to the number of subbands. Fig.4 shows the data distribution onto 4 PE ($jp = 1$) from level $j = 0$ to 2 (where the date redistribution takes place between level 0 and level 1).

Investigations concerning the optimal border data treatment for the decomposition levels not covered by subband based distribution are given in [25] for the 2-D wavelet packet decomposition and in [56] for the 3-D case. A generalization of the subband based data distribution to an arbitrary number of PEs in presented in [28, 29] and the analytical results are verified on a Cray T3D.

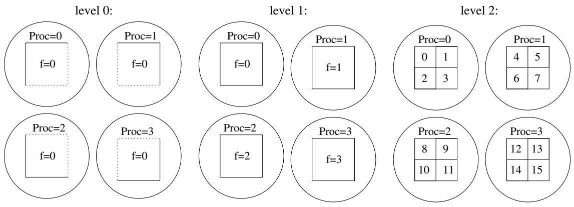


Figure 4. Repartition of the wavelet packets onto 4 PE

The abovementioned approach involving a global transpose instead of border data treatment for the DWT is used also in the context of wavelet packet decompositions – here, it is important to notice that only the highest resolution frequency subbands are generated in this case. This limits this approach to numerical applications where the intermediate frequency subbands are not required [16, 41, 71, 72].

4 Parallel Wavelet Image and Video Coding

Contrasting to the parallel wavelet transform itself, little attention has been paid towards the parallel realization of entire compression applications based on wavelet technology. Wavelet-based compression, possibly applying vector quantization to subbands is implemented on CM-5, CM-200, Symphonie, and SYMPATI2 [73, 74]. Little insight is provided about how the actual compression takes place. Simple thresholding based compression is applied to 1-D signals using non-stationary wavelet packet decomposition in [40], however, nothing is stated about parallel aspects except that domain decomposition is used. Different data access patterns for decomposition and encoding are discussed in [62], experimental results on SGI Origin 2000 of a relatively simple compression scheme consisting of scalar quantization and entropy coding are given employing OpenMP. Two approaches to parallel zerotree wavelet coding are given in [18], an extension to parallel SPIHT coding is discussed in [53]. Different granularity levels for wavelet packet based parallel video encoding are investigated in [31].

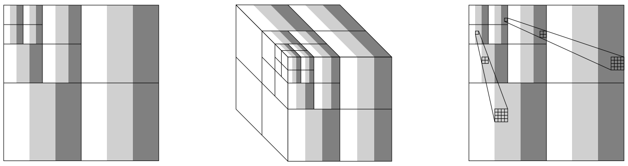
4.1 Parallel SPIHT

Most video compression algorithms rely on 2-D based schemes employing motion compensation techniques. On the other hand, rate-distortion efficient 3-D algorithms exist which are able to capture temporal redundancies in a more natural way (see e.g. [38, 49, 60, 91] for 3-D wavelet/subband coding). Unfortunately, these 3-D algorithms often show prohibitive computational and memory demands (especially for real-time applications). At least, prohibitive for a common microprocessor. A parallel MIMD architecture seems to be an interesting choice for such an algorithm.

Here we concentrate on the parallelisation of the encoding part. As opposed to [18] we will produce a bit-stream that is compatible to the sequential 3-D variant [49] of the SPIHT algorithm [82].

4.1.1 Parallel Wavelet Transform and Zero-Trees

The wavelet filtering is performed in parallel on local data (stripe partitioning in the time domain is used). Before each decomposition step, border data has to be exchanged between neighbouring PEs due to the filter length. After that, transformed data are found distributed as shown in Fig.5.



(a) 2-D case (b) 3-D case (c) Zero-trees

Figure 5. Distribution of coefficients or list entries. Different colours indicate different PEs. (c) shows that zero-trees are local objects.

Zero-tree based algorithms arrange the coefficients of a wavelet transform in a tree-like manner, i.e. each coefficient has a certain number of child coefficients in another subband (mostly 4 in the 2-D, 8 in the 3-D case, see Fig.5(c)).

Furthermore, a zero-tree is a sub-tree which entirely consists of insignificant coefficients. The significance of a coefficient is relative to a threshold which plays an imported role in the SPIHT algorithm ($\mathrm{sig}(c) \Leftrightarrow |c| \geq \mathrm{threshold}$). The statistical properties of transformed image or video data (self-similarity) ensures the existence of many zero-trees. With the help of these zero-trees, sets of insignificant coefficients can be encoded efficiently. We will see that sometimes the root coefficient of the subtree (or even its direct offspring) does not have to be insignificant.

Zero-trees can be viewed as a collection of coefficients with approximately equal spacial position. While this fact implies that the coefficients significances are statistically related which is exploited by the SPIHT algorithm, this also means that zero-trees are local objects corresponding to the data distribution produced by the parallel wavelet transform (see Fig.5(c)). This can be exploited by the parallelisation of the zerotree algorithms.

4.1.2 The SPIHT Algorithm

Although the SPIHT algorithm is sufficiently explained in the original paper [82] it is helpful in this context to reformulate the algorithm.

In the beginning the threshold c is bigger than all the coefficients. Thus, all coefficients are insignificant. Then the threshold is repeatedly divided by 2 and the changes in significance have to be coded. Before and after each step (refinement step) the significance of the coefficients is represented by three lists:

LIS List of insignificant set of pixels. This list includes all roots of zero-trees. An entry in this list can be of two types: Type A – all descendants are insignificant, Type B – all descendants except the direct offspring are insignificant.

LIP List of insignificant pixels. This list includes coefficients that are insignificant but not part of a zero-tree corresponding to an LIS entry.

LSP List of significant pixels.

When processing a refinement step each entry of each list has to be tested for a change of significance and possibly be moved to another list. All entries inserted at the end of a list are also processed in the same refinement step until no more entries are left.

Possible list entry transitions are $\mathrm{LIS} \rightarrow \mathrm{LIS}, \mathrm{LIS} \rightarrow \mathrm{LIP}, \mathrm{LIS} \rightarrow \mathrm{LSP}, \mathrm{LIS} \rightarrow \mathrm{BS}, \mathrm{LIP} \rightarrow \mathrm{LSP}, \mathrm{LIP} \rightarrow \mathrm{BS}, \mathrm{LSP} \rightarrow \mathrm{BS}$ where BS stands for “bit-stream” which means that each evaluation of a list entry has to be coded into the bit-stream (so the decoding process is able to reproduce the decision). Furthermore, when processing the LSP the value of the corresponding coefficient is refined by one bit (which is written into the bit-stream).

The decoding process performs the same algorithm but it does not evaluate the significance of the list entries but simply reads this information from the bit-stream and corrects the value of the corresponding coefficient as good as it can.

4.1.3 SPIHT Parallelisation

When parallelising the SPIHT algorithm we have to face the problem that it uses lists of coefficient positions and is therefore inherently sequential. The basic operations of the algorithm are: Moving an iterator all through a list, deleting elements at iterator position and appending elements at the end of a list. So the aim is to distribute the list so that each PE-local entry corresponds to a local coefficient where coefficients are distributed among the PEs as shown in Fig.5.

For initial distribution this is a simple task but as coefficients are appended to the end of lists one has to provide a mechanism to indicate which parts of a list belong to which

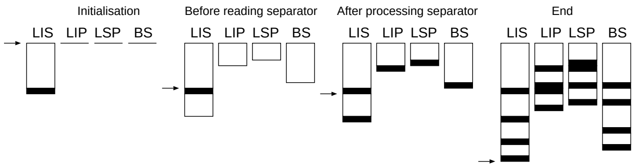


Figure 6. Functionality of separators. Four states of the three lists and the bit-stream while processing the LIS.

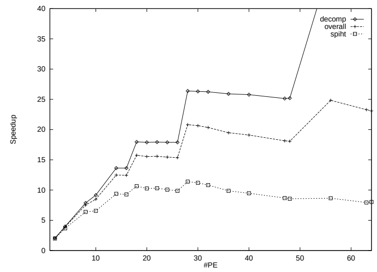


Figure 7. Speedups for varying #PE and fixed compression rate (0.14 bpp).

PE – or from a PEs view: where a sequence of local coefficients ends and parts of another PEs list should be inserted. This work is done by separators (see Fig.6).

The idea is to insert a separator at the end of each part of the list which entirely belongs to a single PE. So the initial distribution is to split the approximation sub-band into equal parts, assigning each part to a list on a single PE and appending a separator to the end of the list. From here on the sequential algorithm is performed locally with one exception: Each time the iterator meets a separator the separator is copied to the end of each destination list. A destination list is a list into which an entry could potentially have been inserted while processing previous iterator positions. Applying this principle the lists L_i on PE_i are split by separators into parts L_{ij} such that the assembled list $L_{11}L_{21}L_{31}\ldots L_{12}L_{22}\ldots$ is identical to the list the sequential algorithm would produce. The same is true for the bit-stream.

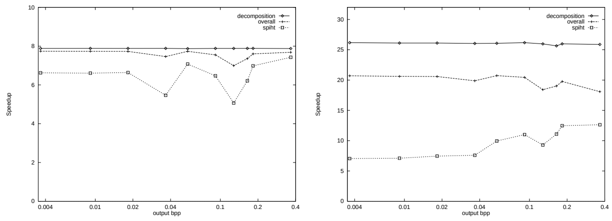
An important question is when the processing of a list is completed. Essentially, the procedure can stop if it has processed the last non-separator entry in the list. Unfortunately, this does not guarantee that each PE produces the same number of separators. But, this is a necessary condition for the correctness of the parallel algorithm because otherwise the correct order of the list-parts would be lost. Therefore, the global maximum number of separators has to be calculated (which unfortunately synchronises the PEs) and the lists have to be filled up with separators before continuing with the next list/refinement step.

The procedure of assembling the bit-stream (after collecting the PE-local bit-streams) is the only sequential part of the algorithm. Unfortunately, it gets more complicated and therefore consumes more calculation time when the number of PEs is increased.

4.1.4 Experimental Results

We present results of an MPI implementation on a CRAY T3E-900 (distributed memory MIMD - multicomputer). Video data size is always 864 frames with 88 by 72 pixels. The video sequence used here is the U-part of “grandma”. The wavelet transform is performed up to a level of 3.

Fig.7 shows speedups for a fixed compression rate: 0.14 bpp (bits per pixel, pixels in different frames are counted as different pixels). The speedup of the SPIHT parallelisation (without wavelet transform) seems to be limited by approximately 10. The reason for this is firstly the increased influence of the sequential part for higher #PE and secondly bigger load balancing problems for higher #PE due to the unevenly distributed complexity in different parts of the image.



(a) 8 PEs (b) 32 PEs

Figure 8. Speedup for decomposition, SPIHT coding and overall speedup for varying compression rate.

Fig.8 shows speedup curves for fixed #PE and varying compression rate. Of course the wavelet decomposition is not dependent on the compression rate. Although the speedup of the coding part (SPIHT) increases with the bpp-value the overall speedup remains constant of drops slightly because the share in execution time of the coding part increases with the bpp-value.

4.2 Parallel JPEG2000

The multiprocessor architecture (i.e. shared memory MIMD) – often also denoted SMP – is an interesting alternative to multicomputers for image processing tasks due to the high memory requirements of these applications. Also, the availability of comfortable programming environments for parallel processing on such architectures (e.g. OpenMP, JAVA Threads) is an important aspect. Finally, the excellent prize-performance ratio of Intel-based SMPs makes such systems very popular for many applications involving visual data processing [81]. In this section, we describe a "straight forward" parallelization of a JPEG2000 codec using JAVA threads.

4.2.1 JPEG2000

The JPEG2000 image coding standard [9, 11] is based on a scheme originally proposed by Taubman and known as EBCOT ("Embedded Block Coding with Optimized Truncation" [90]). The major difference between previously proposed wavelet-based image compression algorithms such as EZW [84] or SPIHT [82] is that EBCOT as well as JPEG2000 operate on independent, non-overlapping blocks which are coded in several bit layers to create an embedded, scalable bitstream. Instead of zerotrees, the JPEG2000 scheme depends on a per-block quad-tree structure since the strictly independent block coding strategy precludes structures across subbands or even code-blocks. These independent code-blocks are passed down the "coding pipeline" shown in Fig.9 and generate separate bitstreams. Transmitting each bit layer corresponds to a certain distortion level. The partitioning of the available bit budget between the code blocks and layers ("truncation points") is determined using a sophisticated optimization strategy for optimal rate/distortion performance.

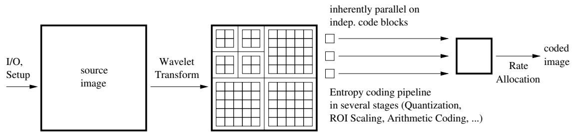


Figure 9. JPEG2000 coding pipeline

The main design goals behind EBCOT and JPEG2000 are versatility and flexibility which are achieved to a large extent by the independent processing and coding of image blocks [9]. The default for JPEG2000 is to perform a five-level wavelet decomposition with 7/9-biorthogonal filters and then segment the transformed image into non-overlapping code-blocks of no more than 4096 coefficients which are passed down the coding pipeline.

In Fig.10 we compare the time required for encoding differently sized images using four image codecs: DCT-based JPEG, wavelet-based SPIHT, Jasper, and JJ2000 (Jasper and JJ2000 both implement the JPEG2000 standard). Note, that JPEG, SPIHT, and Jasper are C/C++ based whereas JJ2000 is written in JAVA (see `http://jj2000.epfl.ch`).

Evidently, JPEG is the by far fastest algorithm, whereas both JPEG2000 implementations are slowest. Interestingly,

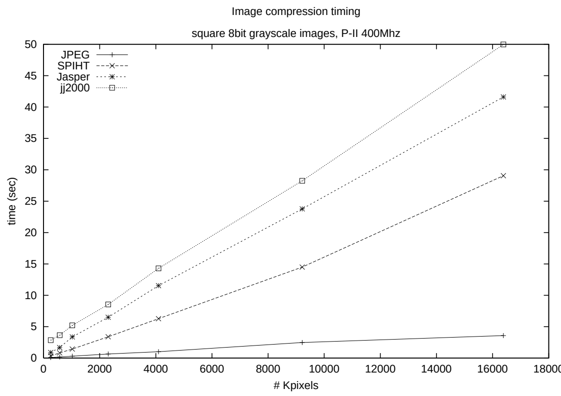


Figure 10. Compression timings

there is not much difference between the C and JAVA implementations (the IBM JDK 1.1.8 just-in-time compiler is used for JJ2000). Fig.11 shows a runtime analysis of the sequential execution of JJ2000. The wavelet transform part is clearly the most demanding part of the algorithm, followed by the encoding stage. Fortunately, both stages can be parallelized with little effort. Intrinsically sequential parts of the algorithm are image and bitstream I/O and R/D allocation which all show relatively low complexity. Obviously, high parallelization potential was one of the design goals of JPEG2000.

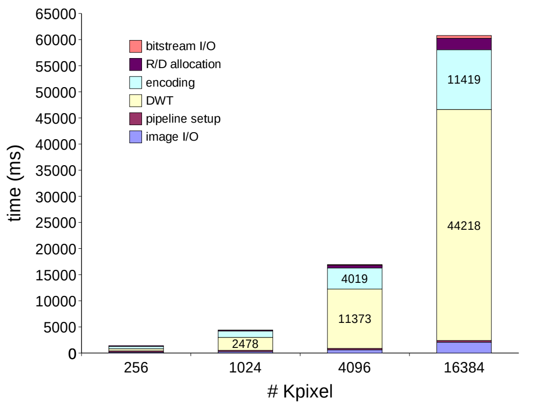


Figure 11. Serial Runtime Analysis of JJ2000 on Intel Pentium II Xeon, 500 MHz

4.2.2 Parallelization of JJ2000 using JAVA Threads

The approach followed in this work is to change as little as possible in the original JJ2000 code for parallelization. JAVA multi-threading is employed in the wavelet transform and encoding stage. For a multi-threaded wavelet transform, different parts of the data are assigned to different threads, the deterministic workload allows a static load allocation. However, synchronization is required at each decomposition level between vertical and horizontal filtering. In the encoding stage, on the other hand, no synchronization is necessary due to the processing of independent code-blocks. The load balance problem caused by the different runtime for each code-block is solved by using a pool of worker threads and a staggered round robin assignment of the code-blocks to these threads. Whereas the JJ2000 code already contains the necessary thread invocation calls for a parallel encoding stage, the transform part is covered in this work.

Fig.12 displays the runtime analysis of a multi-threaded execution on a 4 processor SMP system (a Compaq server with Intel Pentium II Xeon processors running at 500 MHz which is used for all subsequent experiments). An overall speedup of 1.75 is achieved only. When analyzing the chart in more detail, we find that the speedup corresponding to the encoding stage is about 3.6 whereas the wavelet transform speedup is 1.6 at most. Therefore, we investigate the wavelet transform part in more detail.

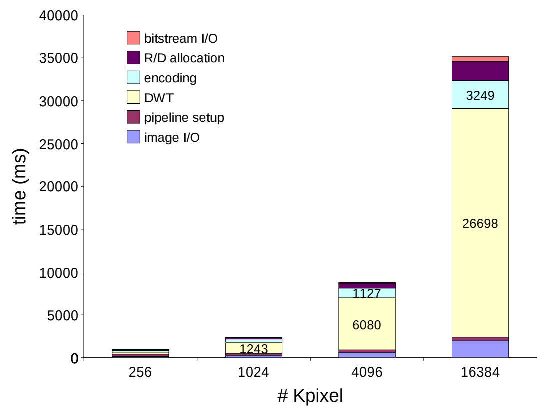


Figure 12. Parallel Runtime Analysis of JJ2000 on SMP (four processor Compaq, Intel Pentium II Xenon, 500 MHz)

Fig.13 shows the timings for the filtering procedures, broken down into the vertical and horizontal parts, respectively. The vertical filtering step requires more than 10 times the execution time of the horizontal counterpart. Surprisingly, also the speedup for the vertical filtering is significantly lower than this for the horizontal case (compare Fig.14).

This unexpected behaviour suggests the existence of a severe cache-miss problem (see also [54] for similar effects in an MPI implementation for a 3-D wavelet decomposition). In fact, it turns out that when using large images

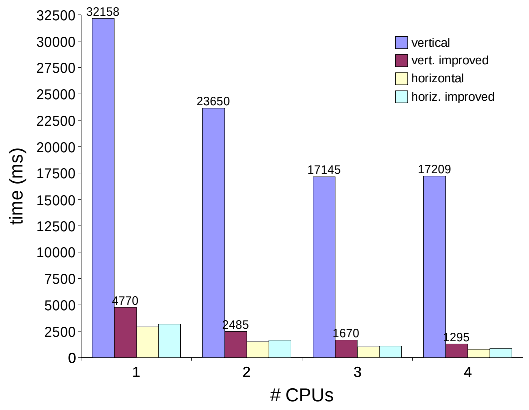


Figure 13. Original and improved filtering

with width equal to a power-of-two and the filter length is longer than 4 (this corresponds to the 4-way associative cache), an entire image column is mapped onto a single cache-set. Consequently, during the execution of vertical wavelet filtering an enormous amount of cache misses occur. We have considered two approaches to improve the cache hit rate. First, the image width is forced to be not a power-of-two (e.g. by inserting dummy data, compare [54]). This technique does not require any modification in the filter code and results in the use of more cache sets and consequently allows cache hits on vertically adjacent pixels. Second, several adjacent columns are filtered concurrently within a single processor. When loading the first data points of an image column into the cache, the corresponding data of adjacent columns are situated within the same cache line. Therefore, computing the products of pixels and filter coefficients of all these columns can be performed without any cache misses (except the initial access which triggers the cache load). Here, a modification of the filter code is required – the results of the different columns have to be buffered. The second approach has turned out to be more effective.

A significant improvement is observed in Fig.13 – almost factor 10 is gained by our technique, horizontal and vertical filtering are now almost identical with respect to runtime. Additionally, the speedup of the improved vertical filtering routine is significantly higher (Fig.14) and now equals that of horizontal filtering. Note that the constrained speedup of the original filtering routine is due to the congestion of the bus caused by the high number of cache misses.

Finally, Fig.15 shows the runtime analysis of JJ2000 with the improved filtering routine. We notice an overall speedup of 5.39 with respect to the original JJ2000 implementation (see Fig.11). Of course, the superlinearity is due to the improved filtering routine. A further significant increase of parallel efficiency can not be expected, since the

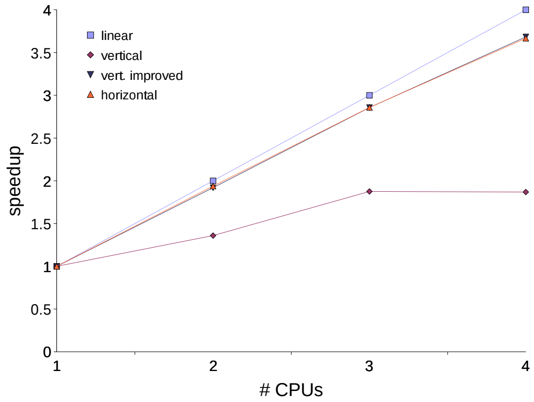


Figure 14. Speedup of filtering routines

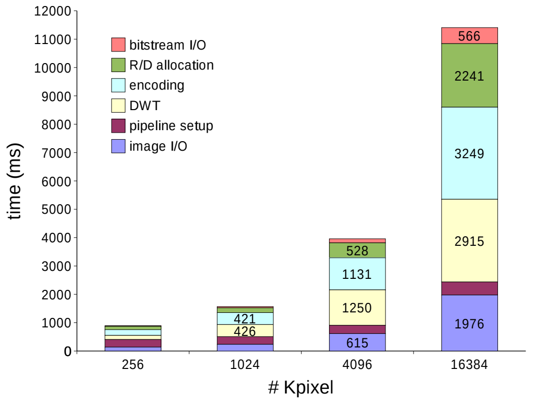


Figure 15. Parallel Runtime Analysis of JJ2000 with improved filtering

intrinsically sequential stages contribute already about 40% to the overall execution time and the efficiency of the parallel parts can hardly be improved without massively changing the code, which is not the scope of this work.

Acknowledgements

This work has been partially supported by the Austrian Science Fund FWF, projects no. P13903 INF and P13732 MAT.

References

[1] S. Akramullah, I. Ahmad, and M. Liou. A data-parallel approach for real-time MPEG-2 video encoding. *Journal of Parallel and Distributed Computing*, 30:129–146, 1995.

[2] S. Akramullah, I. Ahmad, and M. Liou. Performance of software-based MPEG-2 video encoder on parallel and distributed systems. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(4):687–695, 1997.

[3] S. Akramullah, I. Ahmad, and M. Liou. Software-based H.263 video encoding using a cluster of workstations. In H. Shi and P. Coffield, editors, *Parallel and Distributed Methods for Image Processing*, volume 3166 of *SPIE Proceedings*, pages 266–273, 1997.

[4] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies. Image coding using wavelet transform. *IEEE Transactions on Image Processing*, 1(2):205–220, 1992.

[5] A. Averbuch, D. Lazar, and M. Israeli. Image compression using wavelet transform and multiresolution decomposition. *IEEE Trans. on Image Process.*, 5(1):4–15, 1996.

[6] S. Bevinakoppa and other. Implementation of JPEG algorithm on SHIVA parallel architecture. In V. Prasanna, V. Bhaktar, L. Patnaik, and S. Tripathi, editors, *Parallel Processing – Proceedings of the International Workshop on Parallel Processing*, pages 184–188. Tata-McGraw Hill, 1994.

[7] A. Bruckmann, T. Schell, and A. Uhl. Evolving subband structures for wavelet packet based image compression using genetic algorithms with non-additive cost functions. In *Proceedings of the International Conference "Wavelets and Multiscale Methods" (IWC'98), Tangier, 1998*. INRIA, Rocquencourt, Apr. 1998. 4 pages.

[8] C. Chakrabarti and M. Vishvanath. Efficient realizations of the discrete and continous wavelet transforms: From single chip implementations to mappings on SIMD array computers. *IEEE Transactions on Signal Processing*, 3(43):759–771, 1995.

[9] M. Charriera, D. S. Cruz, and M. Larsson. JPEG2000, the next millenium compression standard for still images. In *Proceedings of the IEEE ICMCS'99*, June 1999.

[10] P. Cheng, J. Li, and C. Kuo. Rate control for an embedded wavelet video coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(7):696–702, 1997.

[11] C. Christopoulos, A. Skodras, and T. Ebrahimi. The JPEG2000 still image coding system: an overwiew. *IEEE Transactions on Consumer Electronics*, 46(4):1103–1127, 2000.

[12] C. Chu. Genetic algorithm search of multiresolution tree with applications in data compression. In H. Szu, editor, *Wavelet Applications*, volume 2242 of *SPIE Proceedings*, pages 950–958, 1994.

[13] C. Chui, editor. *Wavelets: A Tutorial in Theory and Applications*. Academic Press, San Diego, 1992.

[14] R. Coifman and M. Wickerhauser. Entropy based methods for best basis selection. *IEEE Transactions on Information Theory*, 38(2):719–746, 1992.

[15] G. Cook and E. Delp. An investigation of scalable SIMD I/O techniques with application to parallel JPEG compression. *Journal of Parallel and Distributed Computing*, 30:111–128, 1996.

[16] S. Corsaro, L. D'Amore, and A. Murli. On the parallel implementation of the fast wavelet packet transform on MIMD distributed memory environments. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 357–366. Springer-Verlag, 1999.

[17] P. Cosman, R. Gray, and M. Vetterli. Vector quantization of image subbands: A review. *IEEE Transactions on Image Processing*, 5(2):202–225, 1996.

[18] C. Creusere. Image coding using parallel implementations of the embedded zerotree wavelet algorithm. In B. Vasudev, S. Frans, and S. Panchanathan, editors, *Digital Video Compression: Algorithms and Technologies 1996*, volume 2668 of *SPIE Proceedings*, pages 82–92, 1996.

[19] B. Das, R. Mahapatra, and B. Chatterji. Modeling of wavelet transform for de bruijn graph connected multiprocessors. In *PDPTA International Conference*, pages 1482–1489, 1997.

[20] P. Desarte, B. Macq, and D. Slock. Signal-adapted multiresolution transform for image coding. *IEEE Transactions on Information Theory*, 38(2):897–904, 1992.

[21] A. Downton. Generalized approach to parallelising image sequence coding algorithms. *IEE Proc.-Vis. Image Signal Processing*, 141(6):438–445, Dec. 1994.

[22] T. El-Ghazawi and J. LeMoigne. Wavelet decomposition on high-performance computing systems. In *Proceedings of the 1996 International Conference on Parallel Processing*, volume 2, pages 19–23, 1996.

[23] J. Falkemeier and G. Joubert. Parallel image compression with JPEG for multimedisa applications. In J. Dongarra et al., editors, *High Performance Computing: Technologies, Methods & Applications*, number 10 in Advances in Parallel Computing, pages 379–394. North Holland, 1995.

[24] M. Feil, R. Kutil, and A. Uhl. Parallel wavelet transforms on multiprocessors. In P. Amestoy et al., editors, *Parallel Processing. Proceedings of EuroPar'99*, volume 1685 of *Lecture Notes on Computer Science*, pages 1013–1017. Springer-Verlag, 1999.

[25] M. Feil and A. Uhl. Algorithms and programming paradigms for 2-D wavelet packet decomposition on multicomputers and multiprocessors. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 367–376. Springer-Verlag, 1999.

[26] M. Feil and A. Uhl. Real-time image analysis using wavelets: the "à trous" algorithm on MIMD architectures. In D. Sinha, editor, *Real-Time Imaging IV*, volume 3645 of *SPIE Proceedings*, pages 56–65, 1999.

[27] M. Feil and A. Uhl. Real-time image analysis using wavelets: the "à trous" algorithm on MIMD architectures. *IS&T/SPIE's Electronic Imaging International Technical Group Newsletter*, 9(2):4–5, 1999.

[28] M. Feil and A. Uhl. 2-D Wavelet Packet decomposition on multicomputers. In *Proceedings of the 8th Euromicro Workshop on Parallel and Distributed Processing*, pages 351–356. IEEE Computer Society, 2000.

[29] M. Feil and A. Uhl. Multicomputer algorithms for wavelet packet image decomposition. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'2000)*, pages 793–798, Cancun, Mexico, 2000. IEEE Computer Society.

[30] M. Feil and A. Uhl. Real-time image analysis using MIMD parallel "à trous" wavelet algorithms. *Real-time Imaging*, 2001. To appear.

[31] M. Feil and A. Uhl. Wavelet packet based video coding in parallel and distributed environments. In S. Panchanathan, V. Bove, and S. Sudharsanan, editors, *Media Processors 2001*, volume 4313 of *SPIE Proceedings*, 2001.

[32] M. Feil, A. Uhl, and M. Vajtersic. Continous wavelet transform on massively parallel arrays. In E. D'Hollander, G. Joubert, F. Peters, U. Trottenberg, and R. Völpel, editors, *Parallel Computing: Fundamentals, Applications and New Directions*, number 12 in Advances in Parallel Computing, pages 207–210. North Holland, 1998.

[33] M. Feil, A. Uhl, and M. Vajtersic. Computation of the continuous wavelet transform on massively parallel SIMD arrays. *Parallel Processing Letters*, 9(4):453–466, 1999.

[34] J. Fridman and E. Manolakos. Distributed memory and control VLSI architectures for the 1-D discrete wavelet transform. *IEEE VLSI Signal Processing*, (10), 1994.

[35] J. Fridman and E. Manolakos. On the scalability of 2D discrete wavelet transform algorithms. *Multidimensional Systems and Signal Processing*, 8(1–2):185–217, 1997.

[36] J. Froment and S. Mallat. Second generation compact image coding. In [13], pages 655–678. Academic Press, 1992.

[37] I. Gertner, R. Peskin, and S. Walther. Parallel computation of the continous wavelet transform. In *Adaptive Signal Processing*, volume 1565 of *SPIE Proceedings*, pages 414–422, 1991.

[38] K. Goh, J. Soraghan, and T. Durrani. New 3-D wavelet transform coding algorithm for image sequences. *Electronics Letters*, 29(4):401–402, 1993.

[39] E. Goirand, M. Wickerhauser, and M. Farge. A parallel two-dimensional wavelet packet transform and some applications in computing and compression analysis. In R. Motard and B. Joseph, editors, *Applications of Wavelet Transforms in Chemical Engineering*, pages 275–319. Kluwer Academic Publishers Group, 1995.

[40] C. Guerrini and D. Lazzaro. Parallel deconvolution and signal compression using adapted wavelet packet bases. In E. Hollander, G. Joubert, F. Peters, and D. Trystram, editors, *Parallel Computing: State of the Art and Perspectives*, volume 11, pages 617–624. 1996.

[41] C. Guerrini and M. Piraccini. Parallel wavelet-galerkin methods using adapted wavelet packet bases. In C. Chui and L. Schumaker, editors, *Approximation Theory VIII: Wavelets and Multilevel Approxiamtion*, pages 133–142. 1995.

[42] J. Hämmerle and A. Uhl. Fractal image compression on MIMD architectures II: Classification based speed-up methods. *Journal of Computing and Information Technology (Special Issue on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia)*, 8(1):71–82, 2000.

[43] Y. He, I. Ahmad, and M. Liou. Modeling and scheduling for MPEG-4 based video encoder using a cluster of workstations. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 306–316. Springer-Verlag, 1999.

[44] T. Hopper. Compression of gray-scale fingerprint images. In H. Szu, editor, *Wavelet Applications*, volume 2242 of *SPIE Proceedings*, pages 180–187, 1994.

[45] T. Huntsberger and B. Huntsberger. Hypercube algorithm for image decomposition and analysis in the wavelet representation. In Walker and Stout, editors, *Proceedings of the 5th Distributed Memory Conference*, pages 171–175. IEEE Computer Society Press, 1990.

[46] D. Jackson and W. Mahmoud. Parallel pipelined fractal image compression using quadtree recomposition. *The Computer Journal*, 39(1):1–13, 1996.

[47] D. Jackson and G. Tinney. Performance analysis of distributed implementations of a fractal image compression algorithm. *Concurrency: Practice and Experience*, 8(5):357–380, June 1996.

[48] A. Khokhar, P. Thulasiraman, G. Heber, and G. Gao. Load adaptive algorithms and implementations for the 2D discrete wavelet transform on fine-grain multithreaded architectures. In *Proceedings of IPPS/PDPS 1999*. IEEE Press, 1999.

[49] B. Kim, Z. Xiong, and W. Pearlman. 3-D set partitioning in hierarchical trees (3-D SPIHT). *IEEE Transactions on Circuits and Systems for Video Technology*, 8(10):1374–1387, 2000.

[50] S. Kim, S. Rhee, J. Jeon, and K. Park. Interframe coding using two-stage variable block-size multiresolution motion estimation and wavelet decomposition. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(8):399–410, 1998.

[51] C. Koc, G. Chen, and C. Chui. Complexity analysis of wavelet signal decomposition and reconstruction. *IEEE Trans. on Aereospace and Electronic Systems*, 30(3):910–918, July 1994.

[52] D. Krishnaswamy and M. Orchard. Parallel algorithm for the two-dimensional discrete wavelet transform. In *Proceedings of the 1994 International Conference on Parallel Processing*, pages III:47–54, 1994.

[53] R. Kutil. Zerotree based video coding on mimd architectures. In S. Panchanathan, V. Bove, and S. Sudharsanan, editors, *Media Processors 2001*, volume 4313 of *SPIE Proceedings*, 2001.

[54] R. Kutil and A. Uhl. Hardware and software aspects for 3-D wavelet decomposition on shared memory MIMD computers. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 347–356. Springer-Verlag, 1999.

[55] R. Kutil and A. Uhl. Optimization of 3-d wavelet decomposition on multiprocessors. *Journal of Computing and Information Technology (Special Issue on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia)*, 8(1):31–40, 2000.

[56] R. Kutil and A. Uhl. Parallel adaptive 3-d wavelet analysis for fast and efficient video coding. In E. D'Hollander, G. Joubert, F. Peters, and H. Sips, editors, *Parallel Computing: Fundamentals and Applications. Proceedings of the International Conference ParCo 1999*, pages 316–323. Imperial College Press, 2000.

[57] G. Lafruit and J. Cornelius. Parallelization of the 2D fast wavelet transform with a space-filling curve image scan. In A. Tescher, editor, *Applications of Digital Image Processing XVIII*, volume 2564 of *SPIE Proceedings*, pages 470–482, 1995.

[58] E. Lega et al. A parallel algorithm for structure detection based on wavelet and segmentation analysis. *Parallel Computing*, 21:265–285, 1995.

[59] K. Leung, N. Yung, and P. Cheung. Parallelization methodology for video coding – an implementation on the TMS320C80. *IEEE Transactions on Circuits and Systems for Video Technology*, 8(10):1413–1423, 2000.

[60] A. Lewis and G. Knowles. Video compression using 3D wavelet transforms. *Electronics Letters*, 26(6):396–398, 1990.

[61] J. Lu, V. Algazi, and R. Estes. Comparative study of wavelet image coders. *Optical Engineering*, 35(9):2605–2619, 1996.

[62] M. Lucka and T. Sorevik. Parallel wavelet-based compression of two-dimensional data. In A. Handlovicova, M. Komornikova, K. Mikula, and D. Sevcovic, editors, *ALGORITMY 2000. Proceedings of the 15th Conference on Scientific Computing*, pages 227–235. Slovak University of Technology, 2000.

[63] P. Lukowicz and R. Cober. A massively parallel implementation of the full search vector quantization algorithm. In W. Gentzsch and U. Harms, editors, *High Performance Computing and Networking. Proceedings of HPCN Europe 1994*, volume 796 of *Lecture Notes on Computer Science*, pages 420–421. Springer, 1994.

[64] V. Manian and R. Vásquez. Efficient algorithms for discrete Gabor transforms using multicomputer networks. *Proc. SPIE: Signal Processing, Sensor Fusion, and Target Recognition V*, 2755:394–403, June 1996.

[65] M. Manohar and J. Tilton. Progressive vector quantization on a massively parallel SIMD machine with application to multispectral image data. *IEEE Trans. on Image Process.*, 5(1):142–146, 1996.

[66] D. Marpe, H. Cycon, and W. Li. Complexity constrained best-basis wavelet packet algorithm for image compression. *IEE Proceedings Vision, Image, and Signal Processing*, 145(6):391–398, 1998.

[67] S. Martucci, I. Sodagar, T. Chiang, and Y. Zhang. A zerotree wavelet video coder. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(1):109–118, 1997.

[68] F. Meyer, A. Averbuch, and J. Strömberg. Fast wavelet packet image compression. *IEEE Trans. on Image Process.*, 9(5):792–800, May 2000.

[69] M. Misra and T. Nichols. Computation of 2-D wavelet transforms on the CM-2. In *Proceedings of the International Conference on Applications in Parallel and Distributed Computing*, 1994.

[70] M. Misra and V. Prasanna. Parallel computation of 2-D wavelet transforms. In *Proc. of the 11th IAPR Int. Conference on Pattern Recognition*, volume IV, pages 111–114. IEEE Comput. Soc. Press, 1992.

[71] L. B. Montefusco. Parallel numerical algorithms with orthonormal wavelet packet bases. In C. Chui, L. Montefusco, and L. Puccio, editors, *Wavelets: Theory, Algorithms and Applications*, pages 459–494. Academic Press, San Diego, 1994.

[72] L. B. Montefusco. Semi-orthogonal wavelet packet bases for parallel least-squares approximation. *Journal of Computational and Applied Mathematics*, 73:191–208, 1996.

[73] P. Moravie, H. Essafi, C. Lambert-Nebout, and J.-L. Basille. Real-time image compression using data-parallelism. In *Euro-Par'95 Parallel Processing*, Lecture Notes in Computer Science 966, pages 723–726. Springer, 1995.

[74] P. Moravie, H. Essafi, and M. Pic. Parallel wavelet transform algorithm for image compression. In F. Huck and R. Juday, editors, *Visual Information Processing IV*, volume 2488 of *SPIE Proceedings*, 1995.

[75] H. Nicolas, A. Basso, E. Reusens, and M. Schutz. Parallel implementations of image sequence coding algorithms on the CRAY T3D. Technical Report Supercomputing Review 6, EPFL Lausanne, 1994.
[76] O. Nielsen and M. Hegland. Parallel performance of fast wavelet transforms. *International Journal of High Speed Computing*, 11(1):55–74, 2000.
[77] J. Patel, A. Khokhar, and L. Jamieson. Scalability of 2-D wavelet transform algorithms: analytical and experimental results on coarse-grain parallel computers. In *Proceedings of the 1996 IEEE Workshop on VLSI Signal Processing*, pages 376–385, 1996.
[78] M. Pic and H. Essafi. Wavelet transform on Connection Machine and SYMPATI 2. *International Journal of Modern Physics C*, 4(1):97–103, 1993.
[79] A. Pommer. Fractal video compression on shared memory systems. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 317–326. Springer-Verlag, Feb. 1999.
[80] K. Ramchandran and M. Vetterli. Best wavelet packet bases in a rate-distortion sense. *IEEE Trans. on Image Process.*, 2(2):160–175, 1993.
[81] C. Rothlübbers and R. Orglmeister. Parallel image processing using a Pentium based shared-memory multiprocessor system. In H. Shi and P. Coffield, editors, *Parallel and Distributed Methods for Image Processing*, volume 3166 of *SPIE Proceedings*, pages 46–54, 1997.
[82] A. Said and W. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *IEEE Transactions on Circuits and Systems for Video Technology*, 6(3):243–249, 1996.
[83] T. Schell and A. Uhl. Customized evolutionary optimization of subband structures for wavelet packet image compression. In N. Mastorakis, editor, *Advances in Fuzzy Systems and Evolutionary Computation*, pages 293–298, Puerto de la Cruz, S, Feb. 2001. World Scientific Engineering Society.
[84] J. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. on Signal Process.*, 41(12):3445–3462, 1993.
[85] K. Shen, G. Cook, L. Jamieson, and E. Delp. An overview of parallel processing approaches to image and video compression. In M. Rabbani, editor, *Image and Video Compression*, volume 2186 of *SPIE Proceedings*, pages 197–208, 1994.
[86] K. Shen, L. Rowe, and E. Delp. A parallel implementation of an MPEG1 encoder: faster than real-time ! In A. Rodriguez, R. Safranek, and E. Delp, editors, *Digital Video Compression: Algorithms and Technologies*, volume 2419 of *SPIE Proceedings*, pages 407–418, 1995.
[87] I. Sodagar, H. Lee, P. Hatrack, and Y. Zhang. Scalable wavelet coding for synthetic/natural hybrid coding. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(2):244–254, 1999.
[88] S. Sullivan. Vector and parallel implementations of the wavelet transform. Technical report, Center for Supercomputing Research and Development, University of Illinois, Urbana, 1991.
[89] C. Taswell. Satisficing search algorithms for selecting near-best bases in adaptive tree-structured wavelet transforms. *IEEE Transactions on Signal Processing*, 44(10):2423–2438, 1996.
[90] D. Taubman. High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, 9(7):1158 – 1170, 2000.
[91] D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 5(3):572–588, Sept. 1993.
[92] P. Topiwala, editor. *Wavelet Image and Video Compression*. Kluwer Academic Publishers Group, Boston, 1998.
[93] A. Uhl. Adapted wavelet analysis an moderate parallel distributed memory MIMD architectures. In A. Ferreira and J. Rolim, editors, *Parallel Algorithms for Irregulary Structured Problems*, volume 980 of *Lecture Notes in Computer Science*, pages 275–284. Springer, 1995.
[94] A. Uhl. Vector and parallel wavelet transforms for the analysis of time varying signals. In R. Schreiber et al., editors, *Proceedings of the seventh SIAM conference on parallel processing for scientific computing*, pages 9–14, 1995.
[95] A. Uhl. Image compression using non-stationary and inhomogeneous multiresolution analyses. *Image and Vision Computing*, 14(5):365–371, 1996.
[96] A. Uhl. Wavelet packet best basis selection on moderate parallel MIMD architectures. *Parallel Computing*, 22(1):149–158, 1996.
[97] A. Uhl and J. Hämmerle. Fractal image compression on MIMD architectures I: Basic algorithms. *Parallel Algorithms and Applications*, 11(3–4):187–204, 1997.
[98] G. Uytterhoeven, D. Roose, and A. Bultheel. A wavelet toolbox for large scale image processing. In P. Zinterhof, M. Vajtersic, and A. Uhl, editors, *Parallel Computation. Proceedings of ACPC'99*, volume 1557 of *Lecture Notes on Computer Science*, pages 337–346. Springer-Verlag, 1999.
[99] M. Wickerhauser. *Adapted wavelet analysis from theory to software*. A.K. Peters, Wellesley, Mass., 1994.
[100] M.-L. Woo. Parallel discrete wavelet transform on the Paragon MIMD machine. In R. S. et al., editor, *Proceedings of the seventh SIAM conference on parallel processing for scientific computing*, pages 3–8, 1995.
[101] Z. Xiong, K. Ramchandran, and M. Orchard. Wavelet packet image coding using space-frequency quantization. *IEEE Transactions on Image Processing*, 7(6):892–898, June 1998.
[102] L. Yang and M. Misra. Coarse-grained parallel algorithms for multi-dimensional wavelet transforms. *The Journal of Supercomputing*, 12(1-2):99–118, 1998.
[103] N. Yung and K. Leung. Parallelization of the H.261 video coding algorithm on the IBM SP2 multiprocessor system. In *Proceedings of the IEEE International Conference on Algorithms, Architectures, and Applications for Parallel Processing*, pages 571–578. North Holland, 1997.
[104] A. Zemla. Wavelet transforms computing on PVM. In J. W. J. Dongarra, editor, *Parallel Scientific Computing*, volume 879, pages 534–546, Berlin, Heidelberg, New York, Tokyo, 1994. Springer-Verlag.
[105] Y. Zhang and S. Zafar. Motion-compensated wavelet transform coding for color video compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(2):285–296, 1992.
[106] Y. Zhang and S. Zafar. Wavelet-based video compression. In H. Li, S. Sun, and H. Derin, editors, *Video Data Compression for Multimedia Computing*, pages 1–54. Kluwer Academic Publishers Group, 1997.