Parallel Wavelet Transform for Large Scale Image Processing

D. Chaver, M. Prieto, L. Piñuel, F. Tirado
Departamento de Arquitectura de Computadores y Automática
Facultad de C.C. Físicas. Universidad Complutense.
Ciudad Universitaria s/n 28040 Madrid.
{dani02,mpmatias,lpinuel,ptirado}@dacya.ucm.es

Abstract

In this paper we discuss several issues relevant to the parallel implementation of a 2-D Discrete Wavelet Transform (DWT) on general purpose multiprocessors. Our interest in this transform is motivated by its usage in an image fusion application which has to manage large image sizes, making parallel computing highly advisable. We have also paid much attention to memory hierarchy exploitation, since it has a tremendous impact on performance due to the lack of spatial locality when the DWT processes image columns.

Keywords—2-D wavelet transform, image fusion, Cache-aware wavelet.

1 Introduction

Wavelets and the corresponding wavelet transforms have been one of the most important developments in image processing over the last decade. Although the most outstanding success of this technology has been achieved in image and video coding (state-of-the-art standards such as MPEG-4 or JPEG 2000 are based on the discrete wavelet transform), it has also proven to be a valuable tool for a wide variety of applications in many different fields.

Our interest in this transform is motivated by its application to image fusion [1]. This operation usually involves two stages: a preliminary registration step (where images from different sources would be geometrically registered in order to be superimposed) followed by the real merging process. A simple method for this second step could be to take the average of the source images, pixel by pixel. However, along with simplicity comes several undesired side effects. The basic idea behind wavelet-based fusion schemes is to combine the wavelet decompositions of the source images so that the fused image is obtained by taking the inverse transform.

The system has to merge high resolution panchromatic data (for example 10-m resolution) with simultaneously acquired low resolution multispectral data (for example 20-m resolution) [2]. In this paper we have studied how to reduce the computational cost of this system, or rather, its wavelet transform component, which could be very time-consuming despite its algorithmic complexity being proportional to the problem size. Parallel computing is a direct way of speeding up the wavelet transform, given that this application has to manage large image sizes.

Focusing on the parallel discrete wavelet transform, a significant amount of work has already been done for all sorts of high performance computers. However, we should remark that most of the previous research has concentrated on special purpose hardware (from application specific VLSI architectures and DSPs [3][4], to FPGAs [5]) and out-of-date SIMD architectures such as the Connection Machine [6].

We have centered our research on general purpose multiprocessor systems. Work on these kind of systems includes [7], where two different parallel strategies for the 2-D wavelet transform were compared on the IBM SP2 and the Fujitsu VPP3000 systems. However, it is limited to the so called *Standard* decomposition of the 2-D FWT [8]. In [9] a new parallel wavelet transform is presented, where communication is reduced at the cost of changing the wavelet transform semantic (basically, each processor views each data-block as an independent data-set and applies the wavelet transform on this block, independently of other blocks). Other work includes [10], where several strategies for the wavelet-packet decomposition are studied, and [11], where a parallel wavelet-based compression algorithm based on OpenMP is analyzed. This paper builds on [7] and extends it by considering the *Non-standard* form, which is the decomposition employed by the fusion scheme investigated.

In addition to the use of parallelism, it is of great importance to use memory resources efficiently, because, as is well known, the maximum performance obtainable from current microprocessors is mostly limited by the memory access in many applications. In [12] S. Chatterjee has explored several techniques to improve both *Standard* and *Non-Standard* versions. A new algorithm was proposed in which a non-linear layout, called the Morton layout, was used to store the image instead of the default C language row-major layout (column-major in Fortran). This new layout improves both data cache and TLB reuse. The tuning of our algorithm has been done just using a loop *tiling* technique since the non-linear layout makes the parallel implementation difficult. Nevertheless, as will be shown later, the improvement that can be achieved with loop tiling is satisfactory enough.

The rest of this paper is organized as follows. The investigated wavelet transform is illustrated in Section 2. In Section 3 we show how to improve the data locality of the algorithm by means of tiling. The different strategies employed to get a parallel implementation of the 2-D DWT are illustrated in section 4 along with a performance study of the different approaches. Finally, the paper ends with some conclusions.

2 2-D Wavelet Transform

The discrete wavelet transform (DWT) can be efficiently performed using a pyramidal algorithm based on convolutions with Quadrature Mirror Filters (QMF). The wavelet representation of a discrete signal *S* can be computed by convolving *S* with the lowpass filter *H(z)* and highpass filter *G(z)* and downsampling the output by 2. This process decomposes the original image into two subbands, usually denoted as coarse scale approximation (lower band) and detail signal (higher band). This transform can be easily extended to multiple dimensions by using separable filters, i.e., by applying separate 1-D transforms along each dimension. In particular, we have considered two different 2-D versions (see Fig. 1), commonly known as the *Standard* and *Non-standard* decompositions [8].

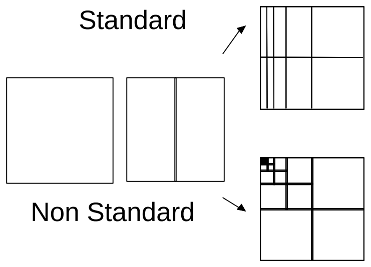


Fig 1. The Standard (top chart) and the Non-Standard (Bottom chart) 2-D DWT algorithms.

The *Standard* decomposition can be easily obtained by performing the complete 1-D DWT on all the rows of the image followed by another 1-D DWT applied to each column as if the transformed rows were themselves an image. In particular, the investigated transform uses Daubechies (9,7) tap biorthogonal filters [13] and symmetric extensions are applied at image boundaries.

The *Non-standard* version alternates between operations on rows and columns, i. e., one stage of the 1-D DWT is applied first to the rows of the image and then to the columns, as shown in Fig. 1. This produces four smaller filtered images and to complete the transformation, the same process is applied recursively to the quadrant containing the coarse scale approximation in both directions. This version is slightly more efficient to compute than the *Standard* decomposition since the data on which computations are performed are reduced to a quarter in each step, as opposed to a half in the *Standard* case.

Although the investigated fusion application employs the *Non-standard* decomposition, for the sake of completeness we have also studied the *Standard* scheme.

3 Memory Hierarchy Optimization

As is well known, the maximum performance obtainable from current microprocessors is mostly limited by memory access. In this context and assuming a row-major layout for the images, as performed by the C language for 2D static arrays, the main bottleneck of the 2-D DWT is caused by the process of image columns.

Among the different techniques studied in literature to improve memory exploitation of 2-D wavelet transforms, one of the most outstanding approach was introduced in [12]. The proposed optimization is based on a non-linear layout of 2D data sets, called the Morton layout, which is shown in figure 2. The running time improvements (due to data cache and TLB reuse) of this layout achieved on a DEC workstation (equipped with a 500 MHz Alpha 21164 microprocessor and 2 MB of L3 cache) reach up to 60% for both *Standard* and *Non-Standard* decompositions [12].

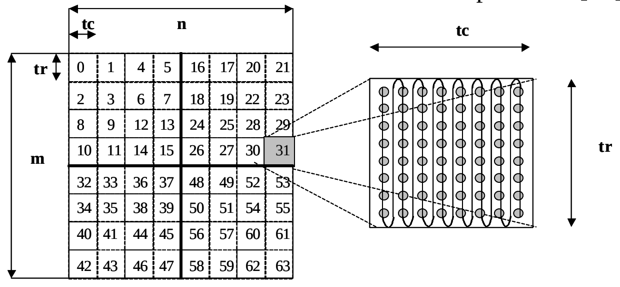


Fig 2. Morton layout.

However, it is difficult to apply this strategy on a parallel setting since this layout makes data decomposition difficult. The tuning of our algorithm has been done just using a loop *tiling* strategy. Instead of processing every image column all the way down in one step, which produces very low data locality (on a row-major layout cache lines are aligned along rows, not along columns) the algorithm is improved by splitting the column processing into a certain number of stages so that spatial locality can be more effectively exploited (see Fig. 3). The benefits of this optimization are shown in Fig. 4. The experimental results have been obtained performing only one level of the wavelet decomposition (i.e. in this case, the *Standard* and the *Non-Standard* versions are equivalent) using different image sizes. As a testing platform we have employed one processor of a SGI

Origin 2000 system equipped with 250 MHz R10000 microprocessors and 4 MB of L2 cache.

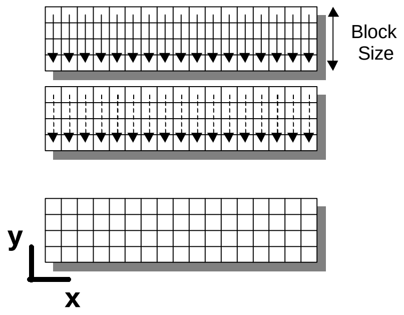
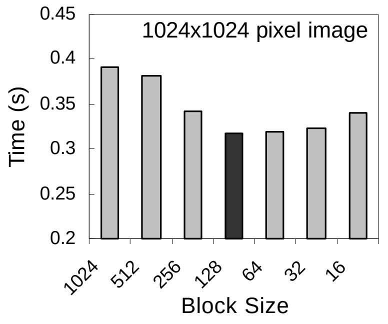


Fig 3. *Loop tiling* optimization. Column processing is split in *num_of_cols/block_size* stages to exploit more effectively data locality.



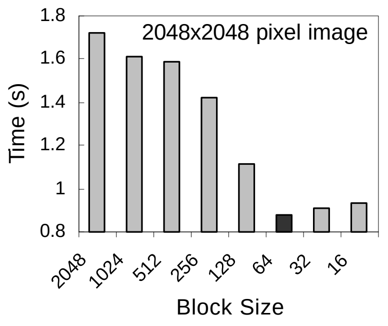


Fig 4. Running time benefits of the *Loop tiling* optimization for 1024^2 (top chart) and 2048^2 (bottom chart) pixel images.

To measure the benefits of the optimization, the performance metric used in [12] is the arithmetic mean over a range of problem sizes of the ratio of the execution time for the optimized code (the one with optimal block size in our case) to the execution time of the non-optimized counterpart. However the sizes employed are not given, which makes a fair comparison impossible. Nevertheless, just to give some numbers, in our code this ratio improves (smaller numbers are better) with problem size and varies from 0.82 for the 1024^2 size to 0.45 for the 4096^2 case. The average ratio reported in [12] for the DEC workstation mentioned above is 0.4. It is our opinion that these results suggest that although column processing could be further improved with a non-linear layout, the optimization achieved by *loop tiling* is satisfactory enough. In addition, given that a non-linear layout makes data decomposition awkward our approach seems more reasonable on a parallel setting.

Due to temporal locality effects, the optimal block size decreases with problem size. In this way, whereas for a 1024^2 image, the optimal block size is around 128 rows, it drops to 64 rows for the 2048^2 case, 32 rows for the 4096^2 problem and so on. Using the hardware counters available on the R10000, we have realized that the main impact of this optimization is on L1 data cache and TLB reuse, whereas the reduction of L2 cache miss is insignificant. For the 2048^2 problem size for example, *loop tiling* almost eliminates TLB misses compared to a TLB miss ratio of almost 1% for the non-optimized version and reduces the L1 miss ratio to 0.6% compared to 5% for the non-optimized code.

From now on in this work, the optimal block size will always be used. We should remark that this optimal size may vary depending on the platform used and the local image size, which is a function of the number of processors and the wavelet level considered.

4 Parallel Wavelet

The most straightforward approach to parallelizing regular applications such as the 2-D DWT using the message passing programming model consists in applying the general principle of domain decomposition, so that each process runs essentially the same program on its share of the data.

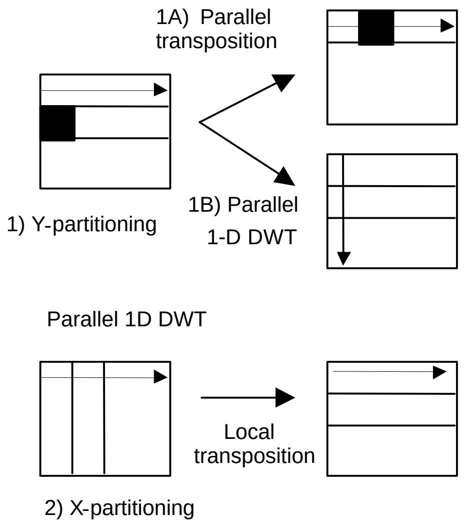


Fig 5. Different data decomposition strategies for the 2-D DWT.

In [7], where just the *Standard* wavelet decomposition is investigated, two different one-dimensional decompositions are considered, which we identified as *X* and *Y* partitioning (see Fig. 5). The *Y-partitioning* approach (denoted as replicated FWT in [7]) , which is the most appealing scheme from an implementational point of view, is based on a parallel image transposition operation such that the 1-D DWT processing is done without any communication. However, the experimental results obtained on an IBM SP2 system showed that the parallel transpose operation dominates the computation time. Therefore, although the *X-partitioning* (called communication-efficient FWT in [7]) approach demands more coding effort (a parallel 1-D DWT is required on row processing), it is the best choice from a performance point of view since it avoids the time-consuming parallel transpose operation.

Guided by these previous results, we have investigated in this work the same *X-partitioning* scheme (apart from some memory optimizations) and a different *Y-partitioning* approach (see Fig. 5) in which we do not use any kind of image transposition at the expense of a parallel 1-D DWT for column processing.

4.1 Impact of Image Transposition

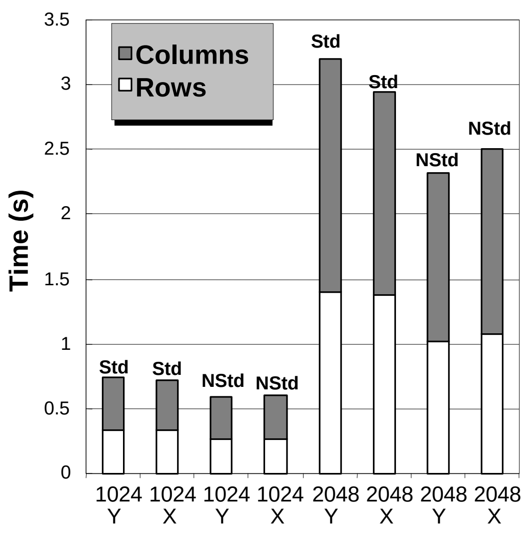


Fig 6. Performance difference of both approaches (X-partitioning versus Y-partitioning) in a sequential setting.

Fig. 6 shows the performance difference of both approaches on a sequential setting, i.e. just using one SGI Origin 2000 processor, performing the complete wavelet decomposition. The differences between the investigated schemes are caused by the different processing of the image columns. The *X-partitioning* processes image columns in the same way as image rows at the expense of a transposition, in which another *loop tiling* optimization has to be applied (an important point not considered in [7]). In the *X-partitioning* counterpart (the *Y-partitioning*), the transpose operation is avoided and image columns are processed using the *loop tiling* optimization discussed in section 3.

For the *Standard* wavelet decomposition, the best choice is the *X-partitioning* approach (around 7% better than the *Y-partitioning* counterpart almost independently of the image sizes) since the extra cost of the image transpose operation is by far compensated by the efficient processing of all the image columns. For the *Non-Standard* decomposition, the best choice is less obvious, since the *X-partitioning* approach requires in this case twice the number of wavelet levels transpositions (remember that the *Non-Standard* algorithm alternates between operations on rows and columns at every wavelet level). For the investigated images sizes the *Y-partitioning* slightly overcomes the *X-partitioning* by a small 1% to 3%.

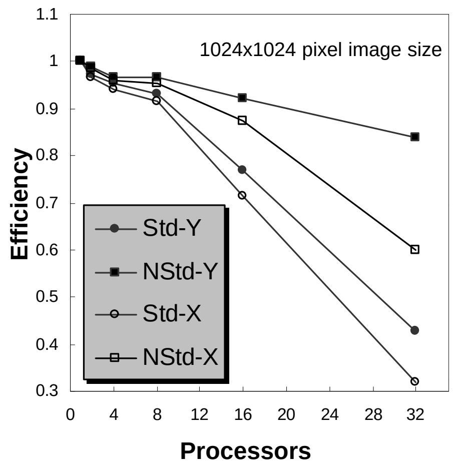
4.2 Replication

Given a certain number of processors, the parallel version is only worthwhile for images larger than some minimal size. Due to the pyramidal structure of the DWT, this means that from some wavelet level, which we have denoted as the *critical level* (that depends on the parallel topology and the number of filter coefficients), a parallel DWT implementation cannot improve the execution time of its sequential counterpart. Indeed, it can deteriorate the performance due to an unsatisfactory communication to computation ratio.

This problem, which is very common in other multi-level algorithms [14], may be alleviated in some cases by setting the number of levels such that the maximum level is the critical one. This approach, which is the strategy considered in [7], is well suited for the investigated 2-D DWT since the need of a complete wavelet decomposition is not usually required in many applications. Indeed, the efficiencies reported in [7] for the *Y-partitioning* scheme applied to the *Standard* wavelet decomposition are quite satisfactory .

We have tried to go further by performing the complete decomposition, given that it represents the worst case from a parallel computing point of view. In particular, the strategy that we have employed in both partitionings to manage the *critical level* problem is based on a data replication operation. In this way, from the critical level all the processes can independently perform the rest of the computation. As Fig. 7 shows, this strategy only achieves satisfactory efficiencies for the *Non-Standard* version. Remember that in the *Standard* decomposition the computation is only reduced by half in each step, which means that more data has to be replicated (i.e. many more communications and replicated computations are required) compared to the *Non-Standard* decomposition.

The comparison between the *X* and *Y-partitioning* will be considered in the next section.



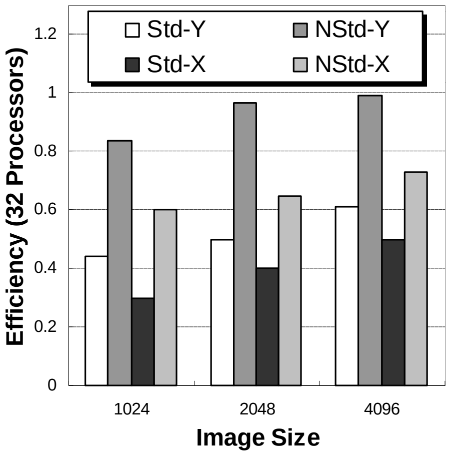


Fig 7. Parallel efficiency of the *Y-partitioning* strategy on the SGI Origin 2000 for both the *Standard and Non-Standard* decompositions (top chart) and parallel efficiency using 32 processors for all the investigated configurations (bottom chart).

4.3 X-partitioning versus Y-partitioning

To analyze the experimental measurements, we have split the running time of the DWT into four different components: the computational cost of processing rows and columns up to the critical level (without communication), the communication overhead up to that level and the time expended on the replicated calculations, which also includes the communication cost required to replicate data.

Fig. 8 compares both approaches for the *Standard* decomposition using 32 processors. It is very clear from this figure that the scalability of the algorithm is limited by the replication. Nevertheless, we should note that this strategy achieves satisfactory results if subcritical levels are not processed (the execution profile in this case is just the same of Fig. 8 without the replication overhead).

Comparing both approaches, the *Y-partitioning* is the best choice (apart from the differences on the image columns) independently of the image size. The main reason behind this fact is that communications arising from the replication phase are more expensive in the *X-Partitioning*. This difference is due to data partitioning since it determines the data structure of the boundaries that will be interchanged between the processors. In the *Y-partitioning*, artificial boundaries (halos) are a set of contiguous in-memory data (as long as our code is written in C). However, in the *X-partitioning*, the data belonging to the interchangeable boundaries are *strided*, i.e. each element is a fixed number of elements away in memory from the preceding and subsequent elements of the halo. As we have previously reported [15], the bandwidth reduction due to *strided* memory access is quite significant in the SGI Origin 2000, which explains the difference.

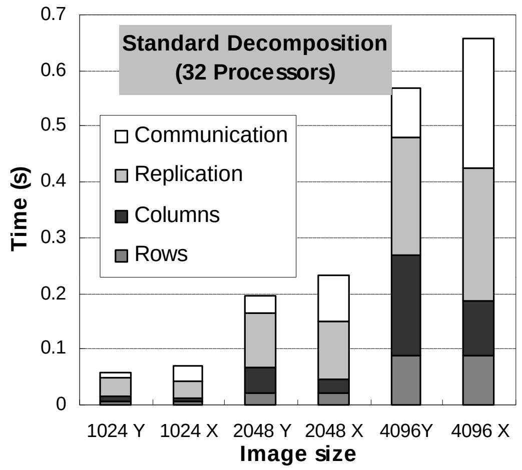


Fig 8. Performance difference between both approaches (X-partitioning versus Y-partitioning) on the Standard decomposition using 32 processors.

Fig. 9 compares both approaches for the *Non-Standard* decomposition. First, it is interesting to note that the parallel efficiency of the *Non-Standard* version is very satisfactory since the extra cost of replication is insignificant. Comparing both approaches, the *Y-partitioning* is again the best choice independently of the image size. In this case, there are two reasons for this result. As we have discussed above, the *X-partitioning* requires twice the number of wavelet levels transpose operations, whose extra cost is not compensated by the row-like processing of all the image columns. On the other hand, communications are again more expensive in the *X-Partitioning* for the same reasons as those explained for the *Standard* version.

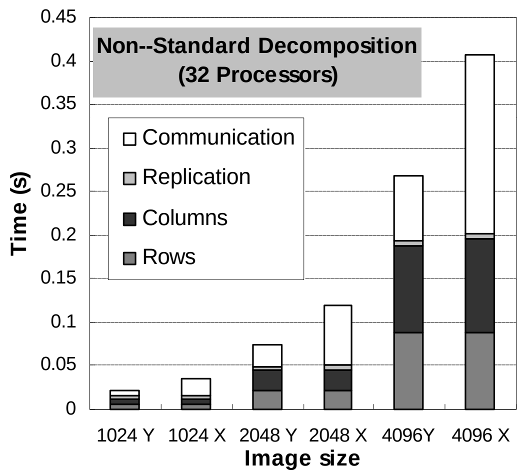


Fig 9. Performance difference of both approaches (*X-partitioning versus Y-partitioning)* on the *Non-Standard* decomposition using 32 processors.

5 Conclusions

In this work, different strategies to get a parallel implementation of the 2-D wavelet transform have been studied. In particular, we have considered two different versions, commonly known as the *Standard* and *Non-standard* decompositions. We should remark that the overall objective of our research is to develop a parallel image fusion application based on the wavelet transform. Although in this paper we have focused on the wavelet transform, future research involves the parallelization of the whole fusion application.

For the *Non-Standard* version, the proposed strategy not only achieves satisfactory efficiencies but also outperforms the scheme employed in [7] due to a lower communication cost. These results also suggest that for this decomposition a major effort should not be made in developing new algorithm where communication is reduced at the cost of changing the wavelet transform semantic [9].

Although the investigated fusion application is based on the *Non-Standard* decomposition, for the sake of completeness we have also studied the *Standard* scheme. In this second case, the data replication required (if the complete wavelet decomposition is applied) limits the scalability of the algorithm. Nevertheless, we should mention that in some situations the complete decomposition is not required and in that case the parallel *Standard* DWT scales satisfactorily [7].

6 Acknowledgments

This work has been supported by the Spanish research grant TIC 99-0474. We would like to thank the *CSC* (Centro de Supercomputación Complutense) for providing access to the SGI Origin 2000 system that has been used in this research.

7 References

[1] Z. Zhang and R. S. Blum. *A Categorization of Multiscale-Decomposition-Based Image Fusion Schemes with a Performance Study for a Digital Camera Application.* Proc. of the IEEE, Vol. 87(8):1315-1325, Aug. 1999.

[2] B. Garguet-Duport, J. Girel, J.M. Chassery and G. Pautou. *The use of Multiresolution Analysis and Wavelet Transform for Merging SPOT Panchromatic and Multispectral Image Data.* Photogrammetric Engineering and Remote Sensing, Vol. 62 (9):1057-1066. Sep. 1996.

[3] M. Martina, G.Masera, G.Piccinini, M.Zamboni *A VLSI Architecture for IWT (Integer Wavelet Transform)* Proc. of 43rd Midwest Symposium on Circuits and Systems, USA, Aug. 2000.

[4] M. A. Trenas. *Arquitecturas y Aplicaciones de la Transformada Wavelet.* Ph. D. Thesis. Dept. de Arquitectura de Computadores, Universidad de Malaga, Nov. 2000.

[5] C. Graves and C. Gloster. *Use of Dynamically Reconfigurable Logic in Adaptive Wavelet Packet Applications.* Proc. of the 5th Canadian Workshop on Field-Programmable Devices, Jun. 1998.

[6] Mats Holmström. *Parallelizing the fast wavelet transform.* Parallel Computing, 11(21):1837-1848, Apr. 1995.

[7] O.M. Nielsen and M. Hegland. *Parallel Performance of Fast Wavelet Transform.* International Journal of High Speed Computing, 11 (1): 55-73, Jun 2000.

[8] A. Fournier. *Wavelet and Their Application in Computer Graphics.* Siggraph' 95 Course notes.

[9] L. Yang and M. Misra. *Coarse-Grained Parallel Algorithms for Multi-Dimensional Wavelet Transforms.* The journal of Supercomputing 11:1-22 , 1997.

[10] M. Feil and A. Uhl. *Multicomputer algorithms for wavelet packet image decomposition.* Proc. of the IPDPS, pp. 793-798, Mexico, 2000.

[11] M. Lucka and T. Sorevik. *Parallel Wavelet-Based Compression of Two-Dimensional Data*. Proceedings of Algorithmy pp. 1-10, 2000.

[12] S. Chatterjee, V. V. Jain, A. R. Lebeck, S. Mundhra and M. Thottethodi. *Nonlinear Array Layouts for Hierarchical Memory Systems.* Proc. of ACM ICS, *pp.* 444-453, Greece, Jun 1999.

[13] I. C. Daubechies. *Ten Lectures on Wavelets.* Philadelphia: SIAM, 1992.

[14] M. Prieto, R. Montero, D. Espadas, I. M. Llorente and F. Tirado. *Parallel multigrid for anisotropic elliptic equations.* Journal of Parallel and Distributed Computing, Academic Press, 61:96–114, Jan. 2001.

[15] M. Prieto, I. M. Llorente and F. Tirado. *Data Locality Exploitation in the Decomposition of Regular Domain Problems.* IEEE Trans. on Parallel and Distributed Systems, 11(11):1141-1149, Nov. 2000.