Hardware and Software Aspects for 3-D Wavelet Decomposition on Shared Memory MIMD Computers

Rade Kutil[1] and Andreas Uhl[2]

[1] Department of Computer Science and System Analysis
University of Salzburg, AUSTRIA
[2] RIST++ & Department of Computer Science and System Analysis
University of Salzburg, AUSTRIA

e-mail: {rkutil,uhl}@cosy.sbg.ac.at

Abstract. In this work we discuss hardware and software aspects of parallel 3-D wavelet/subband decomposition on shared memory MIMD computers. Experimental results are conducted on a SGI POWERChallenge GR.

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, 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 amount of bandwidth required for transmission or storage.

Unfortunately many compression techniques demand execution times that are not possible using a single serial microprocessor [13], which leads to the use of general purpose high performance computers for such tasks (beside the use of DSP chips or application specific VLSI designs). In the context of MPEG-1,2 and H.261 several papers have been published describing real-time video coding on such architectures [1, 3].

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 in the context of the MPEG-4 and JPEG2000 standardization process.

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. [10, 6, 5, 15] for 3-D wavelet/subband coding). Unfortunately these 3-D algorithms often show prohibitve computational and memory demands

(especially for real-time applications). At least, prohibitive for a common microprocessor. A shared memory MIMD architecture seems to be an interesting choice for such an algorithm.

As a first step for an efficient parallel 3-D wavelet video coding algorithm the 3-D wavelet decomposition has to be carried out (followed by subsequent quantization and coding of the transform coefficients). In this work we concentrate ourselves on the decomposition stage.

A significant amount of work has been already done on parallel wavelet transform algorithms for all sorts of high performance computers. We find various kinds of suggestions for 1-D and 2-D algorithms on MIMD computers (see e.g. [16, 14, 12, 7, 4, 8] for decomposition only and [9, 2] for algorithms in connection with image compression schemes). On the other hand, the authors are not aware of any work (except [11]) focusing especially on 3-D wavelet decomposition and corresponding 3-D wavelet based video compression schemes.

In this work we discuss hardware and software aspects of parallel 3-D pyramidal wavelet decomposition on shared memory MIMD computers.

2 3-D Wavelet 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 Figure 1.a).

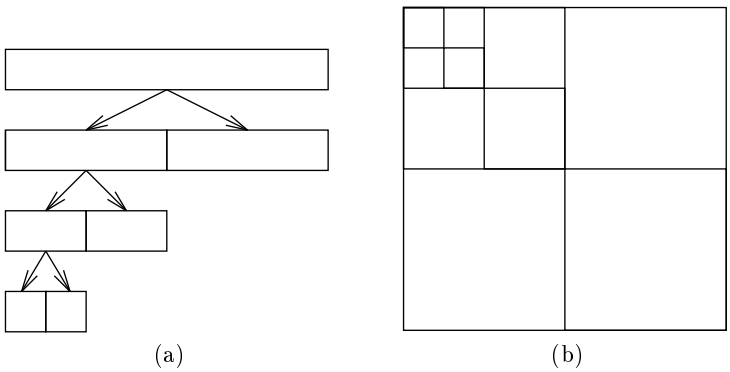


Fig. 1. 1-D (a) and 2-D (b) pyramidal wavelet decomposition.

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 Figure 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.

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 video data. The 3-D data is usually organized frame by frame. The single frames have again rows and columns as in the 2-D case (x and y direction in Figure 2, often denoted as "spatial coordinates"), whereas for video data a third dimension (t for "time" in Figure 2) is added. 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 Figure 2) 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.

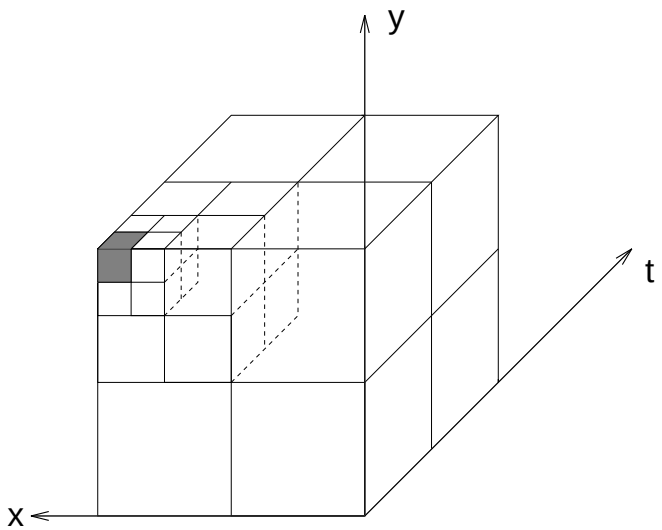


Fig. 2. Classical 3-D wavelet decomposition.

In our implementation we have chosen to apply the very natural frame by frame approach. A pseudo code (using in-place transforms) of such a 3-D wavelet transform with `max_level` decomposition steps applied to a video `S[x,y,t]` looks as follows:

Sequential 3-D Wavelet Decomposition

```
for level=1...max_level {
     for t=1..t_max {
          for x=1...x_max {
               for y=1...y_max {
               convolve S[x,y,t] with G and H } }
          for y=1...y_max {
               for x=1...x_max {
               convolve S[x,y,t] with G and H } } }
     for x=1...x_max {
          for y=1...y_max {
               for t=1..t_max {
               convolve S[x,y,t] with G and H } } }
}
```

3 3-D Wavelet Decomposition on Shared Memory MIMD Computers

3.1 A Message Passing Algorithm

When computing the 3-D decomposition in parallel one has to decompose the data and distribute it among the processor elements (PE) somehow. In contrast to previous work [11] we decompose the data along the time-axis into parallelepipeds (see Figure 3). The obvious reason is that the dimension of the data is expected to be significantly larger in `t` (=time) direction as compared to the spatial directions. This results in better parallel scalability for large machines operating on comparatively small data sets (an important property in real-time processing). Also, the amount of data partition boundaries should be minimized [11] since at these boundaries one has to deal with the border problems of the wavelet transform.

In the literature two approaches for the boundary problems have been discussed and compared [16, 14]. During the *data swapping* method (also known as *non-redundant data calculation*) each processor calculates only its own data and exchanges these results with the appropriate neighbour processors in order to get the necessary border data for the next decomposition level. Employing *redundant data calculation* each PE computes also necessary redundant border data in order to avoid additional communication with neighbour PE to obtain this data. Therefore, redundant data calculation requires a larger initial data set and subsequently trades off computation for communication.

In this work we employ the data swapping approach. Figure 3 shows the stages of the algorithm.

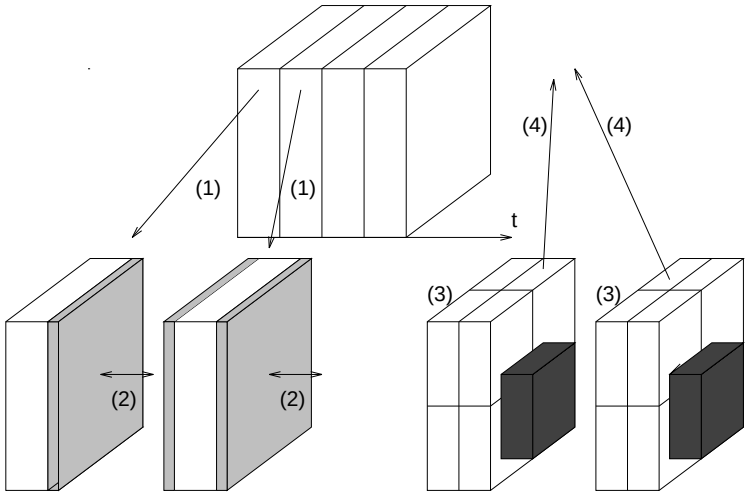


Fig. 3. Message passing with data swapping.

(1) The video data is distributed uniformly among the PE.
(2) The PE exchange the necessary border data (light gray) for the next decomposition level with their corresponding neighbours.
(3) The 3-D decomposition is performed on the local data on each PE.
(4) All subbands but the approximation subband (dark gray) are collected (there is no more work to do on these data).
(5) Repeat steps 2 - 4 until the desired decomposition depth is reached.

Please note that for the first decomposition level steps 1) and 2) are combined by sending overlapping data blocks to the PE. Given p PE and a convolution filter of length f this algorithm shows a computational complexity of order $O(\frac{8}{7}\frac{xyt3f}{p})$ and a transfer complexity of order $O(xy(\frac{2t}{p} + \frac{4(f-2)}{3}))$.

3.2 A Data Parallel Algorithm

Data parallel programming on a shared memory architecture is easily achieved by transforming a sequential algorithm into a parallel one by simply identifying areas which are suitable to be run in parallel i.e. in which no data dependencies exist. Subsequently, local and shared variables need to be declared and parallel compiler directives are inserted.

Since the order of execution is not important and there are no data dependencies among different loop runs (except for the `level` loop) we may apply the following simple parallelization strategy. We distribute the two outer loops `t` and `x` among the PE. Between those two loops the PE need to by synchronized since before the temporal filtering step the spatial filtering has to be completed. Only the indices for the three coordinate axes are declared to be local variables, the data `S[x,y,t]` are of course declared to be shared.

Pseudocode of Data-Parallel Algorithm

```
for level=1...max_level {
#pragma parallel local (x, y, t) shared (S)
#pragma pfor iterate (t,t_max,1)
     for t=1..t_max {
          for x=1...x_max {
               for y=1...y_max {
               convolve S[x,y,t] with G and H } }
          for y=1...y_max {
               for x=1...x_max {
               convolve S[x,y,t] with G and H } }
     endparallelfor
#pragma synchronize
#pragma pfor iterate (x,x_max,1)
     for x=1...x_max {
          for y=1...y_max {
                for t=..t_max {
                convolve S[x,y,t] with G and H } }
     endparallelfor
endparallelregion
}
```

4 Experimental Results

We conduct experiments on a SGI POWERChallenge GR (at RIST++, Salzburg Univ.) with 20 MIPS R10000 processors and 2.5 GB memory. The size of the video data is 128×128 pixels in the spatial domain, combined to a 3-D data block consisting of 2048 frames. The PVM version used as default setting is a native PVM developed for the POWERChallenge Array. We use QMF filters with 4 coefficients.

When performing convolutions on 2-D data using a sensible ordering of the computations modern architectures usually produce not too many cache misses due to the large cache size of the available microprocessors. The situation changes dramatically when changing to 3-D data. In our concrete implementation the spatial wavelet transform (the parallelized `t` loop) is not affected by cache misses since the data is organized frame by frame in the memory. The problem arises when computing the temporal transform (the parallelized `x` loop) since the data involved in this transform is not close in terms of memory location, in contrary, these values are separated by a data block of the size of one frame (this is why prefetching techniques do not work properly as well). Moreover, since we operate with power-of-two sized data, these in-between datablocks are as well of power-of-two size which means that the data values required for the temporal transform are loaded into the same cache lines each time when required within the transform stage (which is quite a few times) which produces a significant amount of cache misses in the transform.

In the original version of our algorithm we result of an sequential execution time of 187 seconds. This execution time can be improved with a simple trick (since it includes many cache misses): instead of having power-of-two sized data in between the data for the temporal transform we insert a small amount artificial dummy data in between the single frames. With this technique we result in an sequential execution time of 82 seconds. Subsequently, all speedup results (despite of those in Figure 4) refer to this improved sequential algorithm.

The cache misses problems do not only affect sequential performance but reduce as well parallel efficiency significantly. Figure 4 shows speedup values of the message passing implementation comparing the original algorithm with the cache corrected version (note that speedup is computed with respect to the corresponding sequential algorithms, to 187 and 82 seconds, respectively). Although the sequential execution time of the original algorithm is much higher (and we would therefore expect a better efficiency) we observe an almost constant speedup for more than 8 PE for the not corrected algorithm. On the other hand, the optimized algorithm shows increasing speedup across the entire PE range. This on first sight surprising behaviour is caused by the cache misses of the not corrected algorithm which lead to a congestion of the bus if the number of PE gets too high.

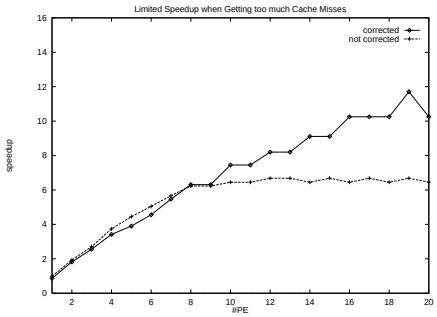


Fig. 4. Reduced efficiency due to cache misses

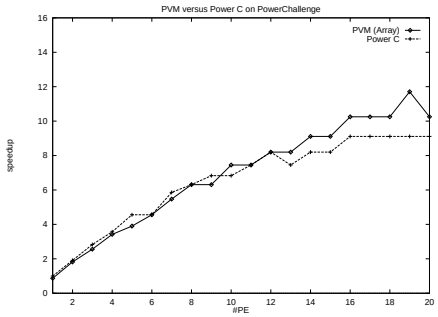


Fig. 5. Comparison of message passing and data parallel algorithms

Figure 5 shows a comparison of the message passing (PVM) and data parallel (PowerC) algorithms. Whereas both algorithms perform almost equally up to 12 PE, the message passing algorithm performs better for higher PE numbers. It should be noted that the message passing implementation uses the host-node paradigm, the number of PE gives in the figure describing the number of node processes being spawned. The actual number of PE in use is therefore one higher. Keeping this is mind we have to compare speedup the speedup at positions PE + 1 (PowerC) and PE (PVM). With this interpretation of the plot the results are different - up to 8 PE actually used the data parallel algorithm exhibits better efficiency, whereas for larger PE numbers the message passing algorithm

still shows better speedup (e.g. we observe speedup 9 versus speedup 12 for 20 PE actually in use.

In order to highlight the importance of an efficient message passing library for satisfying results with this algorithm, we have used as well the public domain version of PVM available at `www.netlib.org` (denoted PVM (Standard) in the plots). We observe a dramatic decrease in efficiency across the entire range of PE numbers. The public domain version of PVM does not allow to reach speedup larger than 2 at all. One fact worth noticing is that for initial data distribution it is not possible to send a single message from the host process to each node since the PVM buffer can not handle data set of this size. Therefore the data is cut into smaller pieces and is sent within a loop.

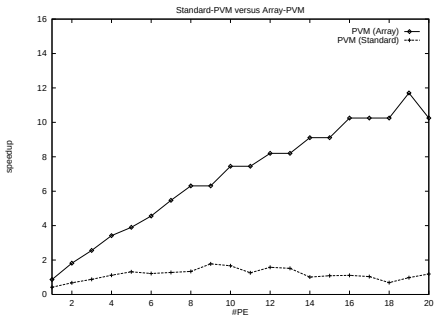


Fig. 6. Comparison of two PVM versions

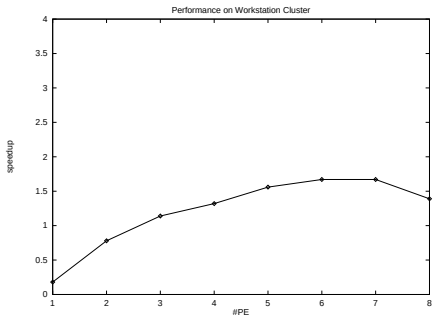


Fig. 7. Performance on a NOW

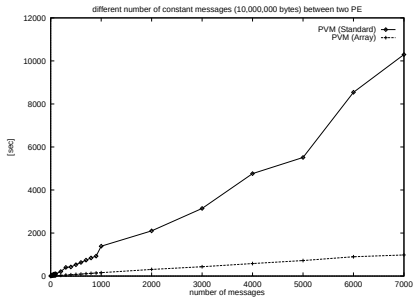
Figure 7 again confirms the former statement about the need for very efficient communication software and hardware. The identical algorithm applied to $64 \times 64 \times 1024$ video data on a FDDI interconnected NOW consisting of 8 DEC/AXP 3000/400 again reaches its speedup maximum at a value of 1.7.

In order to investigate the performance of the two PVM versions on the SGI POWERChallenge in some more detail we measure the effect of sending differently sized messages and of a varying number of messages, respectively.

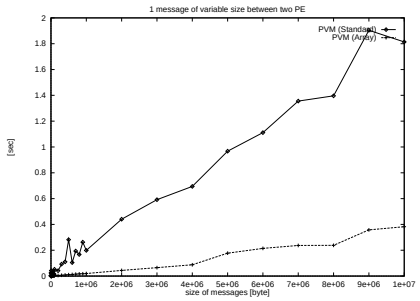
Figure 8 clearly shows that for both cases (large messages and a high number of messages) the native PVM version dramatically outperforms the public domain version.

Having now sufficient information about the scalability of our algorithms applied to a fixed data size, we finally investigate the effect of varying the size of the video data in the spatial (just imagine the difference between QCIF and SHDTV) as well as in the temporal domain (it might be desirable to use (temporal) smaller blocks in order to keep the coding delay to a minimum in a real-time application).

Figures 9 and 10 show that both varying temporal and spatial dimensions do not change the relation between parallel and sequential execution times (since both curves are nearly parallel).



(a) varying number of messages



(b) varying size of one message

Fig. 8. Public domain PVM vs. POWERChallenge Array PVM

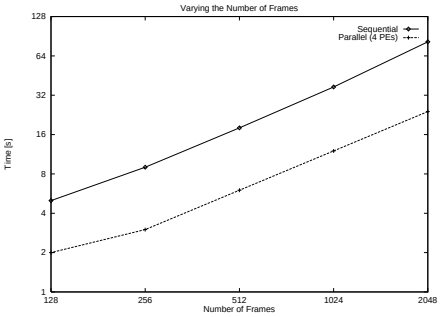


Fig. 9. Temporal scalability

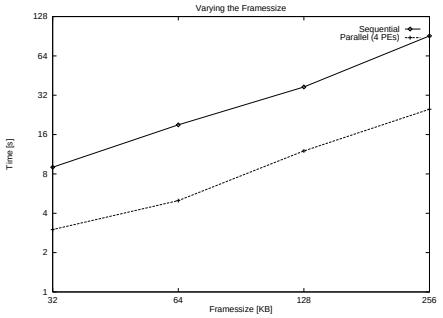


Fig. 10. Spatial scalability

5 Conclusion

In this work we have discussed several aspects of performing 3-D wavelet decomposition on a shared memory MIMD architecture. It has been shown that special attention has to be paid towards cache misses and the right choice of a message passing library. The message passing approach outperforms the data parallel implementation for high PE numbers, whereas we observe the contrary behaviour for small PE numbers.

Acknowledgements

The first author was partially supported by the Austrian Science Fund FWF, project no. P11045-ÖMA. We want to thank Andreas Pommer for his help in resolving the cache problem on the POWERChallenge.

References

1. S.M. Akramullah, I. Ahmad, and M.L. Liou. A data-parallel approach for real-time MPEG-2 video encoding. *Journal of Parallel and Distributed Computing*, 30:129–146, 1995.
2. C.D. Creusere. Image coding using parallel implementations of the embedded zerotree wavelet algorithm. In B. Vasudev, S. Frans, and P. Sethuraman, editors, *Digital Video Compression: Algorithms and Technologies 1996*, volume 2668 of *SPIE Proceedings*, pages 82–92, 1996.
3. A.C. Downton. Generalized approach to parallelising image sequence coding algorithms. *IEE Proc.-Vis. Image Signal Processing*, 141(6):438–445, December 1994.
4. J. Fridman and E.S. Manolakos. On the scalability of 2D discrete wavelet transform algorithms. *Multidimensional Systems and Signal Processing*, 8(1–2):185–217, 1997.
5. K.H. Goh, J.J. Soraghan, and T.S. Durrani. New 3-D wavelet transform coding algorithm for image sequences. *Electronics Letters*, 29(4):401–402, 1993.
6. B.J. Kim and W.A. Pearlman. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPHIT). In *Proceedings Data Compression Conference (DCC'97)*, pages 251–259. IEEE Computer Society Press, March 1997.
7. 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.
8. 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.
9. G. Lafruit and J. Cornelius. Parallelization of the 2D fast wavelet transform with a space-filling curve image scan. In A.G. Tescher, editor, *Applications of Digital Image Processing XVIII*, volume 2564 of *SPIE Proceedings*, pages 470–482, 1995.
10. A.S. Lewis and G. Knowles. Video compression using 3D wavelet transforms. *Electronics Letters*, 26(6):396–398, 1990.
11. 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.
12. J.N. Patel, A.A. Khokhar, and L.H. 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.
13. K. Shen, G.W. Cook, L.H. Jamieson, and E.J. 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.
14. S. Sullivan. Vector and parallel implementations of the wavelet transform. Technical report, Center for Supercomputing Research and Development, University of Illinois, Urbana, 1991.
15. D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 5(3):572–588, September 1993.
16. M-L. Woo. Parallel discrete wavelet transform on the Paragon MIMD machine. In R.S. Schreiber et al., editor, *Proceedings of the seventh SIAM conference on parallel processing for scientific computing*, pages 3–8, 1995.