Optimization of 3-D Wavelet Decomposition on Multiprocessors

Rade Kutil and Andreas Uhl

RIST++ & Department of Scientific Computing
University of Salzburg, AUSTRIA

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

Abstract. In this work we discuss various ideas for the optimization of 3-D wavelet/subband decomposition on shared memory MIMD computers. We theoretically evaluate the characteristics of these approaches and verify the results on parallel computers. Experimental results are conducted on a shared memory as well as a virtual shared memory architecture.

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 bandwidth required for transmission or storage.

Unfortunately many compression techniques demand execution times that are not possible using a single serial microprocessor [17], 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-1,2,4 and H.261 several papers have been published describing real-time video coding on such architectures [1, 5, 2, 3, 8].

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. [14, 9, 7, 19] 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 shared memory MIMD architecture seems to be an interesting choice for such an algorithm.

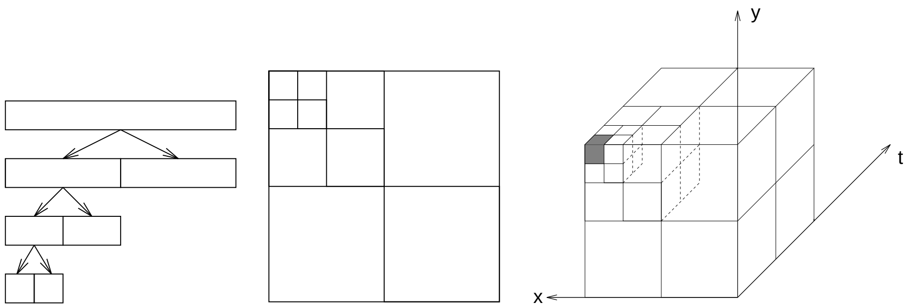
As a first step for an efficient parallel 3-D wavelet video coding algorithm the 3-D wavelet decomposition has to be carried out (followed by subsequent quantization and coding of the transform coefficients). In this work we concentrate on the decomposition stage.

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. [20, 18, 16, 10, 6, 11] for decomposition only and [13, 4] for algorithms in connection with image compression schemes). On the other hand, the authors are not aware of any work (except [15]) 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)).



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

Fig. 1. 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), then convolving the columns of the resulting images with the same filterpair. The same procedure is applied 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 1 (c), often denoted as "spatial coordinates"), whereas for video data a third dimension (t for "time" in Figure 1 (c)) 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.). One decomposition step results in 8 frequency subbands out of which only the approximation data (the gray cube in Figure 1 (c)) is processed further in the next decomposition step. This means that the data on which computations are performed are reduced by $\frac{1}{8}$ in each decomposition step.

In our implementation we have chosen to apply the very natural frame by frame approach. Pseudo code 1 shows, how such a 3-D wavelet transform with `max_level` decomposition steps applied to a video `S[x,y,t]` looks like. We use the notation `a:b` for the sequence $(a, a+1, \ldots, b)$.

```
conv1d (S, G, H, l) :=
    for i in 0:l/2-1
        T[i] = 0; T[l/2+i] = 0
        for j in 0:f-1
            T[i]     = T[i]     + S[2i+j] * G[j]
            T[l/2+i] = T[l/2+i] + S[2i+j] * H[j]
    S = T

for level in 1:max_level
     for t in 0:t_max
          for x in 0:x_max
               conv1d (S[t,x,*], G, H, y_max)
          for y in 0:y_max
               conv1d (S[t,*,y], G, H, x_max)
     for x in 0:x_max+f-1
          for y in 0:y_max
               conv1d (S[*,x,y], G, H, t_max)
```

Pseudo code 1: Sequential 3-D Wavelet Decomposition

Here `f` denotes the length of the filters `G` and `H`. Where the indizes get out of range (for instance `t >= t_max`) `S` should be padded as desired.

Our former experiments [12] show clearly, that special attention has to be paid to the memory organization of the processes and the caches of their processor elements (PE) (see section 4). It is known, that even with common single-processor machines it is a problem to access memory addresses with big distances within short time, because processors are designed to keep chunks of data together in caches. For very large data sets the management of page tables raises similar problems. The optimal use of caches is even more important in parallel computers with shared memory architecture because of the limited bandwidth available for shared memory access (since in the case of cache misses the PE need to access shared memory).

An analysis of the above algorithm shows, that such bad memory accesses happen and cause cache misses [12]. The rearrangement of array indices or memory organization after each convolution of the 3-D array cannot prevent this problem. Pseudo code 2 shows a possibility to rewrite the algorithm completely. It follows the idea to read the input data in a continuous way and to compute immediatly each output or intermediate data, that depends on the input data.

```
revconv (Data, Filter, Coeff, Mod) :=
    l = length (Filter) - 1
    for j in 0:f/2-1
        Data[l-j] = Data[l-j] + Coeff * Filter[2*j+Mod]

for level in 1:max_level
    set C[*,*] and D[*,*,*] to zero
    for t in 0:t_max+f-1
        for x in 0:x_max+f-1
            for y in 0:y_max/2-1
                b = S[t,x,2*y:2*y+f-1] * G
                revconv (C[(x-f)/2+1:x/2,y], G, b, x mod 2)
                revconv (D[(t-f)/2+1:t/2,(x-f)/2+1,y], G,
                         C[(x-f)/2+1,y], t mod 2)
    S = D
```

Pseudo code 2: Optimized Sequential 3-D Wavelet Decomposition

`b` is the value of a convolution of the video data with the filter `G` along the y-axis at `y`. Because `y` is the index of the innermost loop the subsequent computation of `b` can reuse the values loaded into cache before. To understand the procedure `revconv`, we take a look at the non-optimized algorithm. By setting $i' := 2i + j$, $j' := \frac{j}{2}$ and $Mod := j \bmod 2$, $T[i] = T[i] + S[2i + j]G[j]$ can be written as $T[i'/2 - j'] = T[i'/2 - j'] + S[i']G[2 * j' + Mod]$. This is essentially what `revconv` does: It computes all $T[i'/2 - j']$ for a fixed i' (`Coeff` $= S[i']$). `C` holds the results of convolutions along the x-axis. For given `x` and `y` `C[(x-f)/2+1,y]` is completly computed and can be used to compute `D` in the same way. `D` has to be copied into `S` afterwards to continue with the next decomposition step.

For the sake of better readability the algorithm is a bit oversimplified. Products with `H` are not shown at all. So each time `G` is referenced, the same action has to be applied to `H`. The results have to be kept in separate variables (and arrays) and used in succeeding products. Therefore two variables `b` and four arrays `C` (with the size of a quarter of a frame) have to be used producing eight 3-D arrays `D` corresponding to eight subbands. Additionally, setting `C` and `D` to zero can be done within the innermost loop in order to avoid further cache misses. S should be padded as in the non-optimized case.

3 3-D Wavelet Decomposition on Multiprocessors

3.1 Message Passing

We choose a host-node topology for the message passing algorithm. The host PE sends the video data to the node PE and collects the transformed data. The node PE perform all transform computations.

Multidimensional Data Splitting When computing the 3-D decomposition in parallel one has to decompose the data and distribute it among the PE somehow. In previous works parallelization has been done along time axis [12] or in the spatial domain [15]. In this work we use a technique where it can be chosen, how often the video data is split into nearly equal parts in each dimension. In the following p, q and r denote the splitting factors for t, x and y axis respectively (see Figure 2). Of course $pqr \leq \#PE$. The video data is split into parts D_{ijk} where $0 \leq i < p$, $0 \leq j < q$, $0 \leq k < r$ and each D_{ijk} is assigned to PE_l where $l = (iq + j)r + k$. No data is assigned to PE_l with $pqr \leq l < \#PE$.

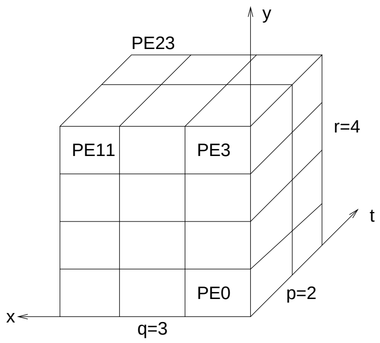


Fig. 2. Example Multidimensional Split of Video Data

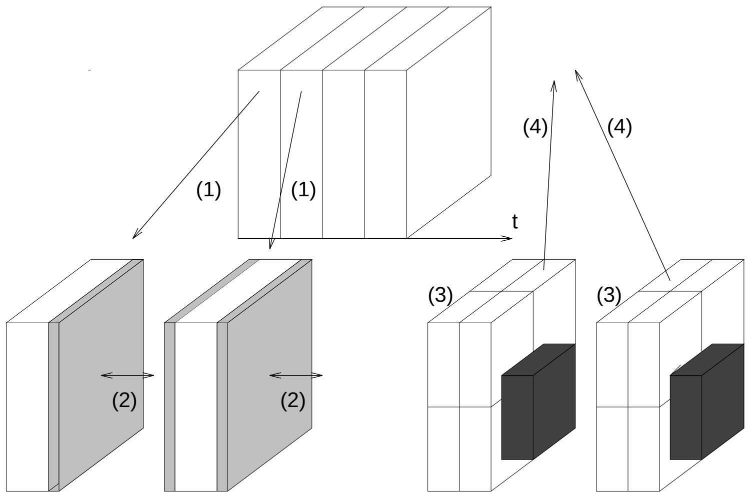
Each D_{ijk} has an approximate size of $\frac{t}{p}\frac{x}{q}\frac{y}{r}$ if t, x, y denote the size of the video data. In a decomposition step overlapping data of a width of $b = f - 2$ (border data) has to be exchanged with neighbouring PE (see Figure 3). Thus border data has a size of $(\frac{t}{p}+b)(\frac{x}{q}+b)(\frac{y}{r}+b)-\frac{t}{p}\frac{x}{q}\frac{y}{r} =: s(p,q,r)$. By minimizing $s(p,q,r)$ with the constraint $pqr = \#PE$ using the Lagrange multiplyer λ we get $\frac{\partial}{\partial(p,q,r)}(s(p,q,r)+\lambda(pqr-\#PE)) = 0$. From this $(p,q,r) = k(t,x,y)$ follows. We have to note two facts: First, if either p, q or r is equal to 1 no border data has to be exchanged along the corresponding axis, thus reducing border data. Secondly p, q and r have to be integers. This leads to the problem that possibly $pqr < \#PE$. But $q = r = 1$ always can guarantee $pqr = \#PE$. These two facts are highly relevant for small #PE and can be overcome by writing a short procedure, that tests all possible p,q,r and minimizes $s(p,q,r)$ where s considers the smaller border data for $p = 1$, $q = 1$ or $r = 1$ and adds a fraction of $\frac{t}{p}\frac{x}{q}\frac{y}{r}$ to get an estimation for the calculation time of one decomposition step depending on p, q and r. The latter portion has to be weighted according to the speed of processors and communication channels.

Combination of Data Swapping and Redundant Data In the literature two approaches for the boundary problems associated with parallel wavelet transform have been discussed and compared [20, 18]. 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.

In this work we combine these methods by performing m steps with redundant data and n subsequent steps with data swapping. $m = 1$ means, that the host PE sends just that amount of redundant data so that the node PE do not have to exchange border data before the first decompostion step. Note that it makes sense to reuse b from the previous paragraph in order to denote the required width of the redundant data ($b = (2^m - 1)(f - 2)$). Here it is even more useful to consider the estimation for the calculation time (see previous subsection).

Figure 3 shows the stages of the algorithm.

Optimization by Partitioning of Source Data Because the last node PE has to wait very long in the start-up phase for its share of the video data, it consequently starts calculation at a later time. Therefore, the processes are executed very asynchronously. Especially when splitting the data in several dimensions as supposed above the exchange of border data can lead to extensive communication that resynchronizes the processes, whereby time is lost. As a solution data can be sent in several (n) parts (`T[t_max*i/n:t_max*(i+1)/n,*,*]` for `i in 0:n-1` if `T` has to be sent to the node PE). The node PE can then perform the according part of the first decomposition step and then wait for the next part to be received.



(1) The video data is distributed uniformly among the PE (including redundant data as necessary).
(2) The PE exchange the border data (light gray) if necessary for the next decomposition level (level $< m$) 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 by the host PE (there is no more work to do on these data).
(5) Repeat steps 2 - 4 until the desired decomposition depth is reached.

Fig. 3. Message passing with data swapping.

3.2 Shared Memory Programming

Program development on a shared memory architecture is easily achieved by transforming a sequential algorithm into a parallel one by simply identifying areas of code which are suitable to be run in parallel i.e. in which few dependencies exist between iterations and different iterations access different data. Subsequently, local and shared variables need to be declared and parallel compiler directives are inserted. In contrast to distributed memory programming there is no need to handle overlapping data regions explicitly.

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 outer loop `t` among the PE. The indices for the three coordinate axes and B are declared to be local variables, the data `S` and other arrays are declared to be shared.

4 Experimental Results

We conduct experiments on an SGI POWERChallenge GR (at RIST++, Salzburg Univ.) with 20 MIPS R10000 processors and 2.5 GB memory and an SGI Origin

```
for level in 1:max_level
    set C[*,*] and D[*,*,*] to zero
#pragma parallel local (t, x, y, B) shared (S,C,D)
{
#pragma pfor iterate (t,t_max+f,1)
    for t in 0:t_max+f-1
        for x in 0:x_max+f-1
            for y in 0:y_max/2-1
                b = S[t,x,2*y:2*y+f-1] * G
                revconv (C[(x-f)/2+1:x/2,y], G, b, x mod 2)
                revconv (D[(t-f)/2+1:t/2,(x-f)/2+1,y], G,
                         C[(x-f)/2+1,y], t mod 2)
#pragma one processor
    S = D
}
```

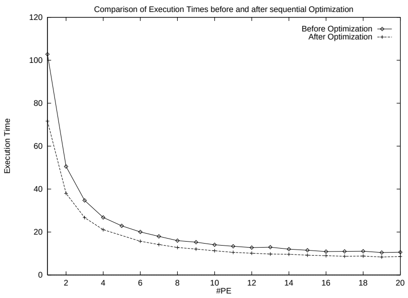
Pseudo code 3: Shared memory implementation

2000 with 30 MIPS R10000 processors (at ZID, Linz Univ.) with 12 GB memory. The size of the video data is 256×256 pixels in the spatial domain, combined to a 3-D data block consisting of 512 frames. QMF filters with 8 coefficients are used. The PVM version employed is a special shared memory variant for SGI systems, shared memory programming is done using SGI PowerC. In all figures except Figure 5 (a) a multidimensional data split of `#PE:1:1` is used, i.e. $p = \#PE$, $q = 1$ and $r = 1$.

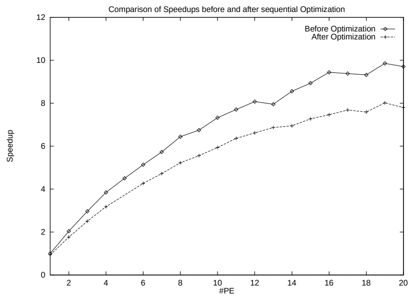
Figure 4 shows the performance improvement we can gain by optimizing the sequential part of the algorithm (applying the message passing algorithm, 2 parallel steps, 1 with redundant data, split along time axis). Of course higher calculation times make communication less important. Therefore the speedup (computed with respect to the corresponding sequential algorithm) is higher in the non-optimized case. In [12] (Fig. 4) a different result was observed for a correction of the sequential algorithm with regard to a problem of excessive cache misses caused by accessing data in large steps with a size of a power of two. The increase in speedup after eliminating undesirable cache misses cannot be achieved again here since the amount of cache misses caused by the algorithm before optimization is too small to congest the bus and therefore does not lead to a reduced efficiency.

To investigate the effect of splitting the video data along several axes we can look at Figure 5 (a). No significant differences may be observed for multidimensional splitting. This is because with small $\#PE$ the size of border data is not that important.

More significant results with respect to partitioning of start data can be seen in Figures 5 (b) and 6. Partitioning of start data synchronizes the execution of processes and prevents the situation that each PE is waiting for the last PE. Figure 6 illustrates two runs with identical parameter sets with and without partitioning of start data. The lowest horizontal line symbolizes the progression

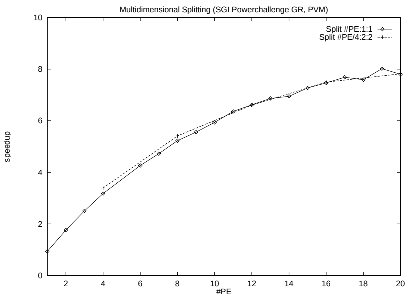


(a) Execution Time

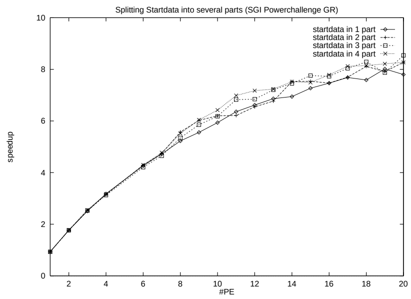


(b) Speedup

Fig. 4. Influence of Sequential Optimization



(a) Multidimensional Splitting

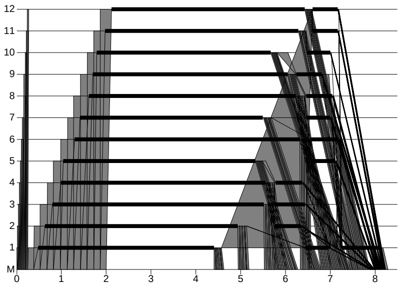


(b) Partitioning of Start Data

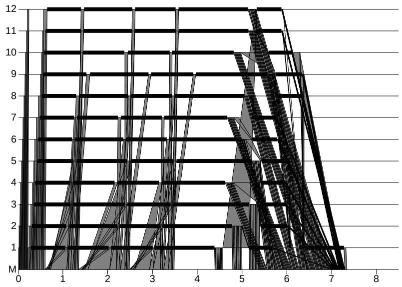
Fig. 5. Optimization of Message Passing Algorithm

of the host PE in time (denoted M), the other represent the node PE. The bold black parts of these time lines represent calculation phases. Crossed lines and gray parts symbolize the transmission of data from one PE to the other, where the gray parts result from the time that is consumed between the beginning and end of a send or a receive. Time is measured in seconds.

It can be clearly observed, that the execution time is reduced by using start data partitioning. Additionally, we observe the significantly different communication structure of the execution (i.e. execution time is reduced although the number of exchanged messages is increased).



(a) without partitioning of start data

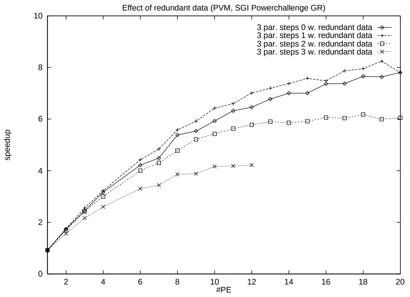


(b) with paritioning of start data

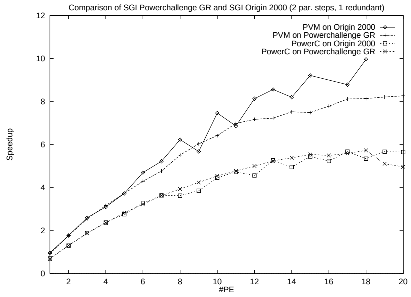
Fig. 6. Execution schemes

The effect of varying the number of decomposition steps that are computed using redundant data can be seen in Figure 7 (a). For one step (i.e. no communication between neighbouring PE just before the first decomposition step) the best speedup is achieved. The use of too much redundant data increases the communication data sizes in the start-up phase as well as calculation times, whereby speedup drops for more than one step in redundant mode.

Another interesting point is the comparison of two physically different architectures: real shared memory (SGI Powerchallenge GR) and virtual shared memory (SGI Origin 2000). Figure 7 (b) shows, that on both machines the shared memory implementation cannot reach the performance of the message passing paradigm. The reason is that when employing the data parallel paradigm, the sequential part of the algorithm operates on shared memory. Because of the excessive memory access (large data) and the associated management of the operating system this slows down the computation. The speedups are about the same on both architectures. One can also observe, that message passing performs better on the virtual shared memory architecture, even though on this machine (SGI Origin 2000) we have a somewhat faster processor and the sequential execution time is about 13% faster. The obvious reason is, that the virtual shared



(a) Varying Redundant Data



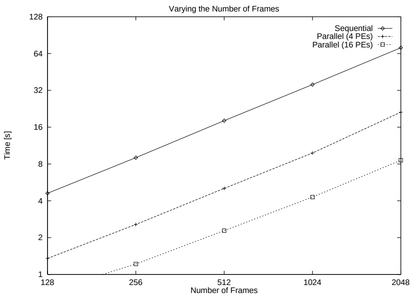
(b) Comparing Architectures

Fig. 7. Redundant Data and Architectures

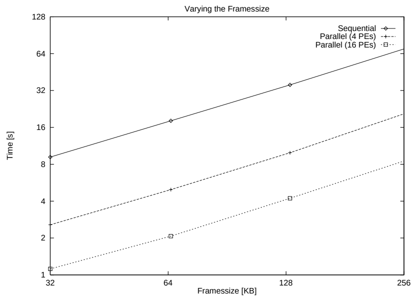
memory architecture provides a more sophisticated structure for interprocessor data exchange.

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).

Figure 8 shows that both varying temporal and spatial dimensions do not change the relation between parallel and sequential execution times (since both curves are almost parallel).



(a) Temporal Scalability



(b) Spatial Scalability

Fig. 8. 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 several optimizations can be applied successfully to the algorithm, while others do not seem to be effective on moderate parallel architectures. Sending redundant data can improve the performance only in the first step. The message passing approach outperforms the data parallel implementation and is additionally able to take advantage of the architecture of virtual shared memory computers.

Acknowledgements

The first author was partially supported by the Austrian Science Fund FWF, project no. P11045-ÖMA. We want to thank the ZID of the University of Linz for providing access to its SGI Origin2000.

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. S.M. Akramullah, I. Ahmad, and M.L. 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.M. Akramullah, I. Ahmad, and M.L. Liou. Parallel MPEG-2 encoder on ATM and ethernet-connected 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 572–574. Springer-Verlag, 1999.
4. C.D. 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.
5. A.C. Downton. Generalized approach to parallelising image sequence coding algorithms. *IEE Proc.-Vis. Image Signal Processing*, 141(6):438–445, December 1994.
6. 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.
7. 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.
8. Y. He, I. Ahmad, and M.L. 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.
9. B.J. Kim and W.A. Pearlman. An embedded wavelet video coder using three-dimensional set partitioning in hierarchical trees (SPIHT). In *Proceedings Data Compression Conference (DCC'97)*, pages 251–259. IEEE Computer Society Press, March 1997.

10. 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.
11. 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.
12. 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.
13. 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.
14. A.S. Lewis and G. Knowles. Video compression using 3D wavelet transforms. *Electronics Letters*, 26(6):396–398, 1990.
15. 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.
16. 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.
17. 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.
18. S. Sullivan. Vector and parallel implementations of the wavelet transform. Technical report, Center for Supercomputing Research and Development, University of Illinois, Urbana, 1991.
19. D. Taubman and A. Zakhor. Multirate 3-D subband coding of video. *IEEE Transactions on Image Processing*, 5(3):572–588, September 1993.
20. 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.