Comparison of PVM and MPI on SGI multiprocessors in a High Bandwidth Multimedia Application

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 the well known wavelet/subband decomposition algorithm (3-D case) widely used in picture and video coding is parallelized using PVM and MPI as well as with data parallel language extensions. The approaches try to take advantage of the possibilities the respective programming interfaces offer. Experimental results are conducted on an SGI POWERChallenge GR and an SGI Origin 2000. These results show a good comparison of the programming approaches as well as the programming interfaces in a practical environment.

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, which all need data compression in order to cope with high memory requirements. 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. Rate-distortion efficient 3-D algorithms exist which are able to capture temporal redundancies (see e.g. [10, 5, 15] for 3-D wavelet/subband coding). Unfortunately these 3-D algorithms often show prohibitive computational and memory demands (especially for real-time applications).

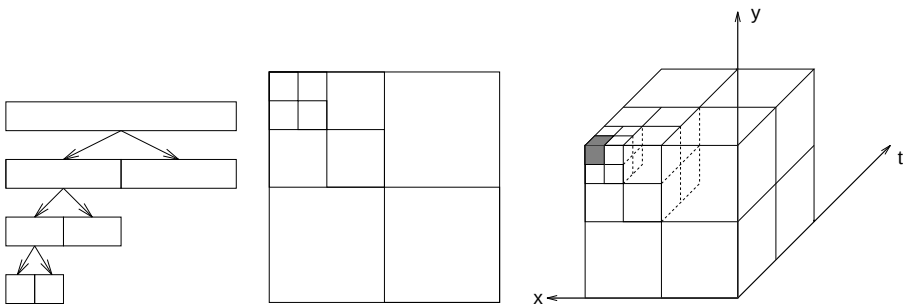
As a first step for an efficient parallel 3-D wavelet video coding algorithm the 3-D wavelet decomposition has to be carried out (followed by subsequent quantization and coding of the transform coefficients). In this work the decomposition step is considered to be the most time consuming and therefore selected for performance comparison.

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, 6, 4, 7] for decomposition only and [9, 2] for algorithms in connection with image compression schemes). On the other hand, few work has been done (except [11]) focusing especially on 3-D wavelet decomposition and corresponding 3-D wavelet based video compression schemes which are very memory consuming.

Therefore it is interesting to see which programming approach is able to cope with large memory and transmission sizes better in an environment of a practical application.

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. 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.

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. 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 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.

Sequential 3-D Wavelet Decomposition - Pseudocode

```
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 Parallelization of 3-D Wavelet Decomposition

3.1 Message Passing

When computing the 3-D decomposition in parallel one has to decompose the data and distribute it among the processor elements (PE) somehow. In previous works data is distributed along the time-axis [8] or along spatial axes [11] into parallel-epipeds (see Figure 2). In this work both variants have been used.

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 to obtain this data.

In this work we employ a combined approach. One can choose, how many of the parallel steps should be processed using redundant data. The succeeding steps then have to swap border data. Figure 2 shows the stages of the algorithm.

(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 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.

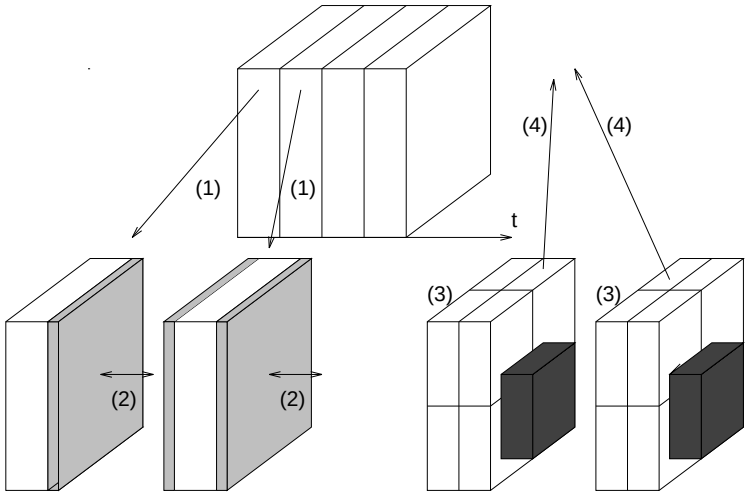


Fig. 2. Message passing with data swapping.

There are several improvements to this scheme:

- One can easily see, that with higher numbers of processors the splitting of the video data produces slices which are so thin, that the border data (that is of the size of the filter length) gets bigger than the slices, which would be very inefficient. One can therefore split data not only in time domain but in spatial domains too, hence distributing subcuboids of data among the PE. This also can reduce the amount of border data that has to be transmitted in each step as a short calculation in section 4 shows.
- It is a fact, that the processes are executed very asynchronously. That is because the last node PE has to wait very long in the start-up phase for its part of the video data, so that it starts calculation at a later time. 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 parts in the beginning. The node PE can then perform the according part of the first decomposition step and then wait for the next part to be received.

3.2 Data Parallel Implementation

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. where different iterations access different data. Subsequently, local and shared variables need to be declared and parallel compiler directives are inserted. This was done with IRIS PowerC using pragma statements.

3.3 Investigation of the abilities of PVM and MPI

It is of great advantage, if processes can continue calculation even if sent data is not yet received by other PEs. The only way to do this with PVM is to provide large communication buffers. On shared memory implementations of PVM it is often possible to specify a shared memory buffer size. This size has to be increased to get good performance and even to avoid deadlocks. Also the transmission of a large data block has to be divided in smaller pieces.

With MPI, the standard `scatter` and `gather` calls provided by MPI could not be used, because of too complex data structures (3-D array). It is not possible to specify one data type that can be used for all subcuboids of the 3-D array, because they do not have equal sizes. Therefore specifying different data types for each data block to be sent/received would have to be necessary, which is not possible even in MPI2.

As to avoid unnecessary copying of data we use non blocking sends and receives instead, and start waiting for their termination at the latest possible time (i.e. after performing calculations which do not affect the data that is still in access). Therewith the hope for improved performance arises.

4 Experimental Results

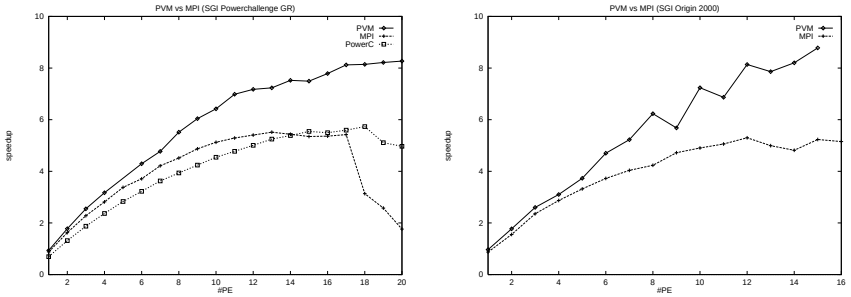
Experiments were conducted on an SGI POWERChallenge GR (at RIST++, Salzburg Univ.) with 20 MIPS R10000 processors and 2.5 GB shared memory and an SGI Origin 2000 (at Linz Univ.) with 30 MIPS R10000 processors. On both IRIX64 release 6.5 is installed. The size of the video data is 256×256 pixels in the spatial domain, combined to a 3-D data block consisting of 512 frames. QMF filters with 8 coefficients were used which is a common size in picture coding applications. The message passing libraries used are SGI-MPI 3.1.1.0 and a special shared memory variant of PVM for SGI systems (SGI PVM 3.3.10).

Figure 3 shows the results of test runs on the two computers mentioned above. The left subfigure additionally shows the speedups for the data parallel implementation with IRIS PowerC.

The relatively poor performance of MPI could not be overcome. A reason for the performance breakdown with high numbers of PEs could not be found. The results for PowerC show, that the data parallel paradigm is not able to reach the performance of message passing.

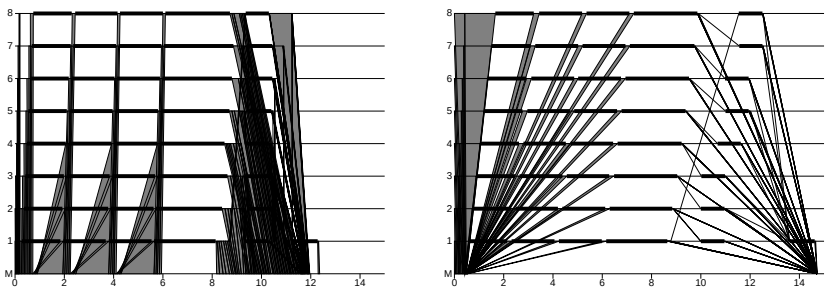
Within Figure 4 two runs with the same parameters with eight PEs are shown, which should illustrate, how the worse performance of MPI is achieved. The lowest horizontal line symbolizes the progression of the master PE in time, the other represent the node PEs. The fat black parts of these time lines 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.

One can see, that there is no algorithmic problem, but each communication step is simply slower than in PVM. Calculation times are not affected.



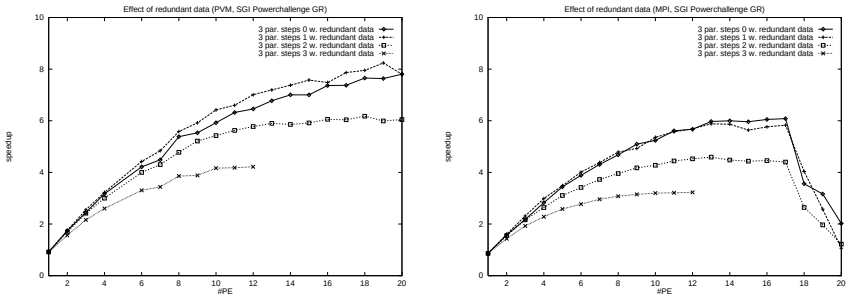
(a) on SGI Powerchallenge GR (b) on SGI Origin 2000

Fig. 3. Comparison: PVM vs MPI



(a) PVM (b) MPI

Fig. 4. Execution scheme of a wavelet decomposition (on SGI PowerChallenge GR)



(a) PVM (b) MPI

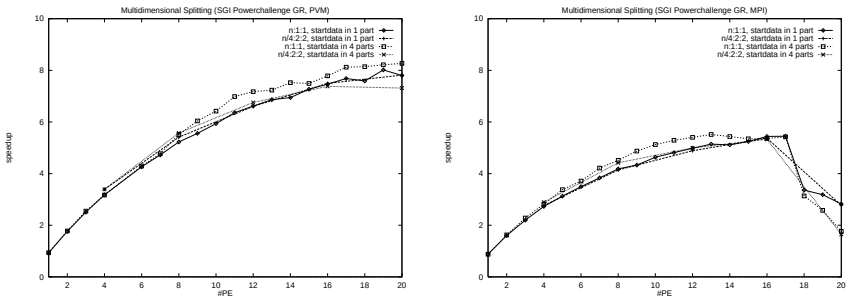
Fig. 5. Redundant data levels

A big question in the topic of parallel wavelet decomposition is, whether redundant data should be used or data swapping as explained in section 3.1. Figure 5 shows, that this question has to be answered differently for PVM and MPI. Where PVM is able to achieve a performance gain by sending redundant data for the first decomposition step, MPI even slows down.

Apart from that, the general behavior is the same. Too much redundant data raises the communication data sizes in the start-up phase as well as calculation times.

The difference in splitting the data along several axes instead of just one can be demonstrated by the following calculation. As we use data of the size of $512 \times 256 \times 256$ pixels, border data normally has a size of $6 \times 256 \times 256$ pixels. 6 is the overhead for a filter length of 8. If we split the data 4 times in time domain and 2 times in spatial domain, one block has the size of $128 \times 128 \times 128$. Border data therefore is $(128+6)^3 - 128^3$, which is about 30% less. But on the other hand, the communication structure gets more complex.

Now, if performance is better in this case, the data size should be the bottle neck. If there are performance differences between PVM and MPI, one could detect characteristics of each of them. But as Figure 6 does not show any significant differences neither between types of data splitting nor between PVM and MPI, no statement can be made. But one can expect different results for systems with lower bandwidth (workstation clusters).



(a) PVM (b) MPI

Fig. 6. Parallelizing along several axes

Acknowledgements

The author was partially supported by the Austrian Science Fund FWF, project no. P11045-ÖMA.

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. 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.
6. 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.
7. 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.
8. R. Kutil and A. Uhl. Hardware and software aspects for 3-D wavelet decomposition on shared memory MIMD computers. volume 1557 of *Lecture Notes on Computer Science*, pages 3347–356. Springer-Verlag, 1999.
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.