Parallel Wavelet Transforms on Multiprocessors*

Manfred Feil, Rade Kutil, and Andreas Uhl

RIST++ & Department of Scientific Computing
University of Salzburg, AUSTRIA
{mfeil,rkutil,uhl}@cosy.sbg.ac.at

Abstract. We discuss several issues relevant for parallel wavelet transforms and their possible implications on the choice of a proper programming paradigm for corresponding multiprocessor implementations.

1 Introduction

In this work we focus onto special problems associated with almost each parallel wavelet algorithm (here we compare pyramidal wavelet decomposition [3], wavelet packet decomposition [2], and the à trous algorithm [1]):

- Data decomposition strategies
- Handling of border data

Specifically we investigate the impact of these problems onto the choice of a proper programming paradigm for multiprocessors (i.e. shared memory programming vs. message passing). It is very interesting to see that very different results occur for different types of algorithms.

2 Wavelet Transform Algorithms

The fast wavelet transform (FWT) can be efficiently implemented by a pair of appropriately designed highpass and lowpass filters. A 1-D wavelet transform of a signal S is performed by convolving S with both filters 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. Higher dimensional FWT is performed in separabel manner leading to $2^s - 1$ detail signals (and one coarse scale approximation) at decomposition level i in the s-dimensional case.

Wavelet packets (WP) represent a generalization of the FWT. Whereas in the wavelet case the decomposition is applied recursively to the coarse scale approximations only, in the wavelet packet decomposition the recursive procedure is applied to all the coarse scale approximations and detail signals, which leads

* This work has been partially supported by the Austrian Science Fund FWF, project no. P11045-ÖMA.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 1013–1017, 1999.

to a complete wavelet packet tree (i.e. binary tree and quadtree in the 1-D and 2-D case, respectively) with 2^{si} frequency subbands at decomposition level i in the s-dimensional case.

The "à trous" algorithm represents a non-orthogonal discrete approach to the classical continuous wavelet transform. The basic idea behind the *à trous* algorithm is to design a discrete wavelet transform *without* a following decimation step. In the 1-D case a filtering operation is performed similar to the computation of the coarse scale approximation in the FWT case. Given the well-known *two-scale equation* found in classical wavelet theory, $\phi(k) = \sqrt{2}\sum_l h_l \phi(2k-l)$, the approximation coefficients are computed by $c_i(k) = \sum_l h_l c_{i-1}(k+l)$ (FWT case) and by $c_i(k) = \sum_l h_l c_{i-1}(k+2^{i-1}l)$ (à trous case). Note that the expression "$(k+2^{i-1}l)$" creates the "trous" (French for holes) in the computation which means that the distance between samples increases by a factor 2 from scale $i-1 \quad (i>0)$ to the next one. This fact has severe implications for possible parallelization approaches at the borders of the data.

In contrast to the FWT the detail signal is computed by $w_i(k) = c_{i-1}(k) - c_i(k)$. À trous decomposition in higher dimension is not performed in separabel manner but by direct convolution with a higher dimensional convolution kernel.

3 Programming Paradigms, Data Decomposition, and Border Treatment

We investigate two different programming paradigms on multiprocessors: shared memory programming and message passing. Shared memory programming can be performed very quickly by simply inserting parallel compiler directives into sequential programs. On the other hand, message passing requires an explicit programming of each communication event occurring among processors and is consequently very time demanding. However, message passing programs written in e.g. MPI or PVM may be used without changes on different architectures (no matter if multiprocessors or multicomputers) whereas shared memory programming mostly uses a native programming language.

The coarse grained programming style required for message passing demands a discussion of data decomposition strategies. Whereas there is nothing to discuss about data decomposition in the 1-D case, different possibilities exist for the 2-D and 3-D cases. The main distinction is between stripe partitioning and checkerboard partitioning (which are the 2-D cases, in the 3-D case simply a third dimension is added). Whereas checkerboard partitioning offers the obvious advantages of minimizing the block-border length at the cost of a larger number of neighbouring blocks, stripe partitioning requires only communication between two direct neighbours.

In the case of wavelet packet decompositions it has turned out that it is advisable to to perform a subband based data decomposition instead of the concepts mentioned before at a certain stage of the computation. This is explained briefly for the 2-D case. To do the decomposition in parallel, the data is redistributed

according to the subband structure (after an initial stripe or checkerboard distribution - see Fig. 1 on the left side) at that specific decomposition level (denoted "distribution level") where the number of PE is lower or equal to the number of subbands. Fig.1 shows the data distribution onto 4 PE from level $j = 0$ to 2 (where the data redistribution takes place between level 0 and level 1 and f denotes the number of the subband).

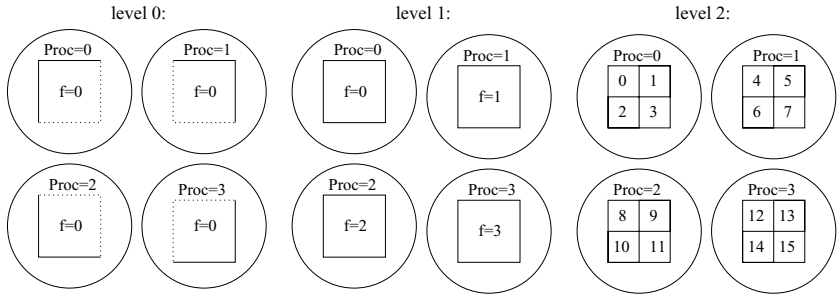


Fig. 1. Repartition of the wavelet packets onto 4 PE

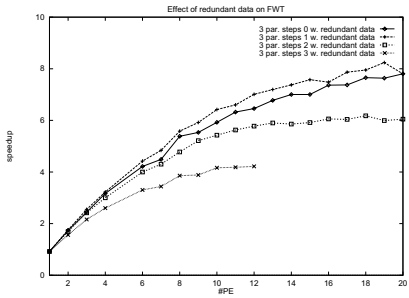
Now we proceed with the discussion of border treatment for all types of wavelet transforms. Recall that the need for border data located on adjacent PE is caused by the nature of the filtering process which involves several neighbouring data points in order to compute a single transform coefficient. In order to provide the necessary border data to each PE we may distinguish between two approaches for border treatment (which trade off computation vs. communication demand):

- *Data swapping* method (also known as *non-redundant data calculation*): each PE computes only non-redundant data and exchanges these results with the appropriate neighbour PE in order to get the necessary data for the next calculation step (i.e. the next decomposition level).
- *Redundant data calculation* approach: in the initialisation step we do not only provide its share of the original signal to a PE but provide the entire data necessary to carry out the required decomposition steps on each PE locally without any communication demand (i.e. a highly redundant data distribution - overlapping blocks).

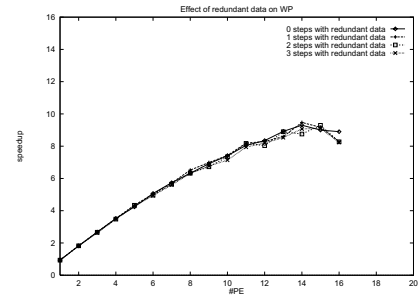
4 Experimental Results

For the 2-D case we employ as test image a 1024×1024 pixel version of the *Lena*-image and perform a complete decomposition (i.e. 10 levels) with Daubechies $W20$ filters. In the 3-D case we use video data with 256×256 pixels per frame and 512 frames. All the computations have been performed on a SGI POWER-Challenge GR with 20 R10000 processors using either the native shared memory programming language PowerC or a native version of the PVM message passing library.

We start with the discussion concerning border treatment in the case of message passing. For 3-D wavelet decomposition (Fig. 2.a) and the 2-D á trous algorithm (Fig. 3.b) data swapping is clearly superior. This may be easily explained by the fact that the amount of redundant computations is too high especially related to the relatively inexpensive communication on the target architecture which is required by data swapping. However, almost no difference occurs in the case of 3-D wavelet packet decomposition (Fig. 2.b) – the obvious reason is that only 3 decomposition steps are performed with redundant data calculation or data swapping, the rest of the computation is performed using subband based data distribution. Therefore, the overall amount of the computations where border problems are involved is rather small and consequently it makes no difference which border-treatment approach to choose. This leads us directly to the question about data decompositions.



(a) 3-D FWT

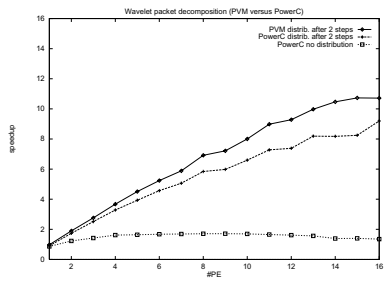


(b) 3-D WP decomposition

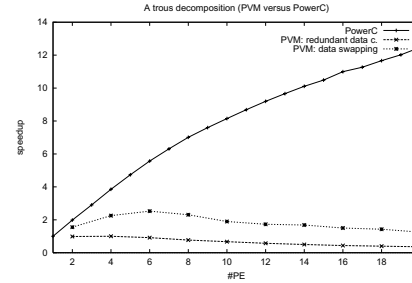
Fig. 2. Data swapping vs. redundant data calculation

Concerning the question whether stripe or checkerboard partitioning is the better way to distribute data it turns out that both methods perform equally on the architecture considered. Fig. 3.a shows the advantage of subband based parallelization for wavelet packet decomposition in a drastic way – almost no speedup is achieved when "no distribution" (i.e. no subband based distribution) is performed, whereas monotonically increasing and significant speedup is achieved with subband based data distribution.

Now let us proceed to the question whether message passing or shared memory programming is the better way for parallel wavelet transforms on multiprocessors. Considering the results for 3-D wavelet packet decomposition (see Fig. 3.a) we clearly see that the algorithm implemented in PowerC exhibits worse scalability as compared to the PVM case. This trend is also observed for 3-D wavelet decomposition (and for both types of algorithms in lower dimension). The PowerC algorithm for 3-D wavelet packet decomposition where simply the loops corresponding to data rows or slices are distributed does not reach any speedup ("no distribution"). If for shared memory programming the subband



(a) WP decomposition



(b) à trous decomposition

Fig. 3. Comparison of programming paradigms: message passing (PVM) vs. shared memory (PowerC)

based data distribution concept is implemented as well (which is fairly complicated to do) we obtain a considerable speedup, but still lower as compared to the message passing case (e.g. for 3-D wavelet packet decomposition speedup 9 vs. 11 with 16 PE).

A very different situation may be observed for the 2-D à trous algorithm (Fig. 3.b). Whereas we do not reach speedup higher than 3 using message passing, speedup close to linear is achieved with shared memory programming. This effect is on the one hand due to the high amount of computation involved in the à trous algorithm (which allows the high speedup), on the other hand due to the expensive border treatment (see section 2 – this causes the bad performing message passing approach).

Whereas shared memory programming obviously is the paradigm of choice for the à trous algorithm and leads to acceptable (but clearly less scalable) results for the FWT, the performance of a straightforward shared memory programming of wavelet packet decomposition is extremely poor. Even if the subband based distribution concept is employed (which requires profund algorithmic knowledge and a significantly higher implementation effort) message passing still remains clearly superior.

References

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

[2] A. Uhl. Wavelet packet best basis selection on moderate parallel MIMD architectures. *Parallel Computing*, 22(1):149–158, 1996.

[3] 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.