Compatibility of Scalapack with the Discrete Wavelet Transform

Liesner Acevedo[1,2], Victor M. Garcia[1], and Antonio M. Vidal[1]

[1] Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, España
{lacevedo, vmgarcia, avidal}@dsic.upv.es
[2] Departamento de Técnicas de Programación
Universidad de las Ciencias Informaticas
Carretera a San Antonio de los Baños Km 2 s/n, La Habana, Cuba

Abstract. Parallelization of the Discrete Wavelet Transform (DWT) oriented to linear algebra applications, specially the solution of large linear systems, is studied in this paper. We propose that for parallel applications using linear algebra and wavelets, it can be advantageous to use directly the two-dimensional block cyclic distribution (2DBC), used in ScaLAPACK [1]. Although the parallel computation of the DWT may be faster with other data distributions, the efficiency of the 2DBC distribution for some linear algebra applications can make it more appropriate. We have tested this approach implementing a preconditioner for iterative solution of linear systems, which uses wavelets and was proposed in [2]. The results using the 2DBC Distribution are better than those obtained with a popular parallel data distribution for computing wavelets, the "Communication-Efficient" scheme proposed in [3].

Keywords: Discrete Wavelet Transform, Parallel Linear Algebra, Scalapack.

1 Introduction

The Discrete Wavelet transform is still a very recent field, so that it is the subject of active research. The number of applications which can benefit from this tool increases nearly every day, such as time series analysis, signal analysis, signal denoising, signal and image compression, fractals, and many more.

The main motivation of this work has been the parallelization of linear algebra algorithms which use wavelets at any stage. There are many papers describing different techniques for the parallelization of the DWT; however, the algorithms described in these papers put the emphasis in computing the DWT as fast as possible; therefore the data distribution among the processors is chosen to minimize the computing time needed to obtain the transformed matrix.

In linear algebra algorithms, the matrix obtained after applying the DWT may have to be used for some calculation far more expensive than the DWT

Y. Shi et al. (Eds.): ICCS 2007, Part I, LNCS 4487, pp. 152–159, 2007.

in itself, such as solution of linear systems. The standard parallel library for linear systems with dense matrices is ScaLAPACK. However, in order to use the ScaLAPACK functions the matrix must be distributed using the bidimensional block cyclic distribution (2DBC distribution), the ScaLAPACK standard. If the matrix is not distributed in this way, one must face either a redistribution of the matrix, (with a high communication cost) or a heavy load imbalance (if no redistribution is done).

We propose to compute the DWT directly over the matrix distributed with the 2DBC distribution. As an application study, we have applied it to the parallelization of a multilevel preconditioner proposed by T. Chan and K. Chen in [2], based on the DWT and the Schur complement method.

The paper starts by giving a small review of the DWT and its application to solution of linear systems. In that section, the multilevel preconditioner is described. Then starts the main part of paper, where the results about parallel DWT are presented. Finally, the code implementing the parallel version of the preconditioner is described, and some numerical results are given.

2 Discrete Wavelet Transform

The theoretical background of the DWT is based on the concept of Multiresolution analysis and is already quite well known. However, these concepts are not useful when considering the implementation details and the parallelization, so that we will refer to the interested reader to the following references: [4,5].

The DWT applied over a signal or vector is obtained through digital filters, a lowpass filter G determined by the coefficients $\{g_i\}_{i=1}^{D}$ and a highpass filter H $\{h_i\}_{i=1}^{D}$.

Not all pairs of highpass and lowpass filters are valid for a DWT; however, there are many such pairs. This means that the DWT is not a single, unique operation, since it depends on the filters. The orthogonal wavelets are the most popular group of wavelets. In this case, to obtain a reversible DWT the filter coefficients must satisfy $h_i = (-1)^i g_i$. The length D of the filter is called the "order" of the wavelet. For simplicity, we shall concentrate only on orthogonal wavelets, and, among these, only on the wavelets depending on short filters, since these will minimize the communications in the parallel DWT.

A single stage of the DWT (or a one-level DWT) over a vector v of length $n = 2^k$ is performed, first, convolving both filters with the vector v, obtaining two sequences of length n. Next, every even element of these sequences is discarded, and the resulting sequences (of length $\frac{n}{2}$) are assembled together.

These two steps are equivalent to the multiplication of the data vector by the matrix $W_n = \begin{pmatrix} H \\ G \end{pmatrix}$, where H and G (of dimension $\frac{n}{2} \times n$) are

$$H = \begin{pmatrix} h_1 & h_2 & \dots & h_D & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & h_1 & h_2 & \dots & h_D & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \ddots & & \ddots & \ddots & \vdots & 0 \\ h_3 & h_4 & \cdots & h_D & 0 & \cdots & 0 & h_1 & h_2 \end{pmatrix} \tag{1}$$

The matrix G has the same structure. This product would result in

$$W_n \cdot v = \begin{pmatrix} H \cdot v \\ G \cdot v \end{pmatrix} = \begin{pmatrix} d_1 \\ \vdots \\ d_{n/2} \\ s_1 \\ \vdots \\ s_{n/2} \end{pmatrix} \tag{2}$$

where the d_i are the high frequency (or "detail") components and the s_i are the low frequency (or "smooth") components. If the wavelet is orthogonal, then the matrix W_n must be orthogonal as well. The orthogonal wavelets with shorter filters are the Haar wavelet, with coefficients $\left(g_1 = \frac{\sqrt{2}}{2}, g_2 = \frac{\sqrt{2}}{2}\right)$

The L-Level DWT (or, simply, the DWT) is obtained applying L times the one-level DWT over the low frequency part of the data vector.

The two-dimensional wavelet transform is applied to a matrix by applying 1-D transforms along each dimension. So, a one-level 2-D DWT applied over a matrix $T_0 \in \Re^{n,n}$ can be written as $\tilde{T_0} = W_n T_0 W_n^T$, with W_n defined as above. In this case, we obtain the following partition:

$$\tilde{T_0} = W_n T_0 W_n^T = \begin{pmatrix} A_1 & B_1 \\ C_1 & T_1 \end{pmatrix}, \tag{3}$$

where $T_1 = GT_0G^T$, $A_1 = HT_0H^T$, $B_1 = HT_0G^T$ and $C_1 = GT_0H^T$.

If, as usual, we want to apply more than one DWT level, at least two different possibilities appear. The so-called "Standard" DWT form of a matrix is obtained applying the L-level DWT over the columns, and then applying the L-level DWT over the rows. The Non-Standard form [6] is obtained applying a one level DWT over the rows, and one over the columns (obtaining a partition as in (3). Then, another one-level DWT by rows ($W_{n/2}^T$) and another DWT by columns ($W_{n/2}$) are applied over the low-frequency submatrix T_1, and this is repeated L times, always over the low frequency submatrix.

When working with matrices, the choice usually is the Non-Standard form or some slightly different version, such as the level-by level form, proposed in [2]. Therefore, all the considerations from now on will be referred to the Non-Standard version.

2.1 Wavelets and Linear Systems

There are several research lines, oriented towards efficient solution of linear systems using wavelets [6,7]. One of these is based on the decomposition shown in equation (3). The main algorithm of interest for us is the recursive Wavelet-Schur preconditioner proposed by Chan and Chen in [2].

It is not possible for us, due to space reasons, to give an appropriate description of the algorithm. Therefore we will only give a brief outline; all the details can be found in [2].

This algorithm uses the block decomposition (3) produced by the DWT. To build a preconditioner, the submatrices A_1, B_1 and C_1 are approximated by "band approximations" (all the elements except those on the desired band are set to zero), and the Schur complement method is used to solve, approximately, a linear system with this new matrix. Combining this technique with standard iterative methods (GMRES, Richardson)[8] a multilevel algorithm for solution of linear systems is built.

The implementation of this algorithm requires the computation of several DWT levels of a matrix, the computation of matrix-vector products of DWT-transformed matrices, and, in the inner level, needs the solution of a linear system through LU decomposition, where the coefficient matrix is again a DWT-transformed matrix.

2.2 Parallel DWT; Data Distributions

The parallel DWT of a matrix is relatively simple to implement [3]. The most critical issue is the data distribution to avoid communications. The scheme with seemingly better properties is the "Communication-efficient" (C-E) DWT proposed by Nielsen and Hegland [3], in which the matrix is distributed by block of contiguous columns.

On the other hand (as mentioned above), if the matrix should be used as coefficient matrix for solution of linear systems, it would be better to use a 2DBC distribution, the used as basis of the Scalapack library. The ScaLAPACK data distribution is described in detail in [1]. Here we only show an example. Let us consider the bidimensional grid of processors of Figure 1.

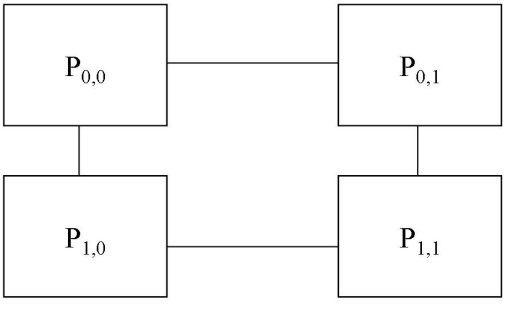

Fig. 1. Bidimensional processor mesh

Figure 2 shows how a matrix would be distributed over the processor mesh in Fig.1. The distribution for C-E DWT is displayed on the left, while the figure on the right shows a possible two dimensional cyclic distribution, suitable for use with Scalapack.

We will examine now which are the communications needed for the computation of a single wavelet level, using both distributions. Let us start with the C-E distribution.

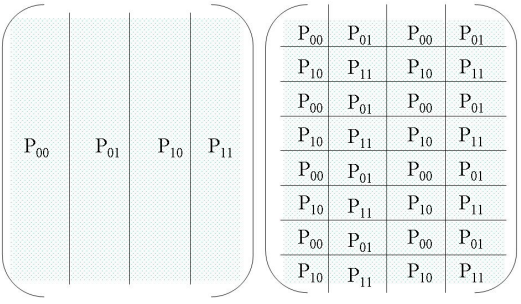

Fig. 2. Matrix on the left, distributed by columns (for C-E DWT); on the right, a possible two-dimensional cyclic distribution

Using basic results from the Nielsen-Hegland paper [3], we have that with the column distribution every processor will need, for each row, $D-2$ elements from the processor holding the columns on the right. These elements are needed to compute the row-oriented 1D DWT. (Since in this distribution each processor holds complete columns, there is no need of communications for the column-oriented 1D DWT). The only exception is the last processor (P_{11} in the figure), which must receive these elements form the first processor (P_{00} in the Figure).

Therefore, considering all rows, each processor will need $N \cdot (D-2)$ elements, and the total number of words to be transferred is $P \cdot N \cdot (D-2)$, where P is the number of processors.

Using a 2DBC distribution, the communications are somewhat more involved. Let us assume first that the matrix is distributed over a square processor mesh of diameter $\sqrt{P}$, and that the block size, N_b divides exactly the dimension of the matrix n. So,if $k = \frac{N}{N_b}$, then the total number of blocks in the matrix is k^2.

Let us consider now a single block, located, for the example, in the processor P_{00}. Using the same result from [3] mentioned above, we obtain that to complete the DWT of that block, the processor P_{00} should receive $N_b \cdot (D-2)$ elements from the processor P_{01} (needed for the row-oriented DWT) and another $N_b \cdot (D-2)$ elements from the processor P_{10} (for the column-oriented DWT). If we consider the last row of blocks, these should be receiving the same amount of data, but from the processor(s) holding the first row of blocks, like in the CE distribution. If we consider the last column of blocks, it would happen exactly the same.

It turns out that every block (no matter in which processor are they located) requires a communication of $2N_b \cdot (D-2)$ words . Therefore, for a single DWT level, the total number of words to be transferred shall be $k^2 \cdot 2N_b \cdot (D-2) = 2N \cdot k \cdot (D-2)$. Clearly, for "normal" values of P and N_b, more communications are needed with the 2DBC distribution.

Of course, if we were interested only in computing the DWT, it would not be sensible to consider a Scalapack-like distribution, since it is simpler and more efficient to compute the DWT with the communication-efficient strategy. Nevertheless, the situation changes when, as in this case, some linear algebra tasks must be performed in parallel with the transformed matrices.

3 Example of Application: Parallel Implementation of the Wavelet-Schur Preconditioner

The Wavelet-Schur preconditioner, described briefly above, is a good example of algorithm which might benefit of the use of the ScaLAPACK library. It needs to compute several matrix-vector products and, above all, it needs to solve a dense linear system.

To test the performance of the data distributions considered, we have written a parallel linear system iterative solver, based on GMRES and preconditioned with the Wavelet-Schur preconditioner.

The GMRES method for dense matrices in a 2DBC distribution is implemented easily using the PBLAS and ScaLAPACK functions for the matrix-vector product, saxpy update, dot product and 2-norm: **pdgemv**, **pdaxpy**, **pddot** and **pdnorm2**. In the coarsest level where the direct method is used, the LU decomposition of ScaLAPACK **pdgesv** is used.

The band approximations of the A, B and C matrices have been chosen as diagonal matrices.

The linear systems used to evaluate the parallel algorithm proposed were generated with the following matrices:

$$a_{i,j} = \begin{cases} 2 & if\ i = j \\ \frac{1}{|i-j|} & otherwise \end{cases} \tag{4}$$

For simplicity, the sizes of the matrix have been chosen as powers of two: $768, 1152, \cdots, 4024$. All the tests were carried out in a cluster with 20 2GHz Intel Xeon biprocessors, each one with 1 Gbyte of RAM, disposed in a 4x5 mesh with 2D torus topology and interconnected through a SCI network. The processor grids used have sizes 2×2 , 3×3 and 4×4 and the block size was 32.

3.1 Numerical Results

The table 1 and 2 shows the CPU times of the algorithm in sequential and parallel versions, using 1 and 2 wavelet levels in the proposed processor meshes with the 2DBC Distribution.

The same algorithm was implemented as well computing the parallel DWT with the CE distribution. It would have been interesting to apply directly the algorithm for solution of linear systems to the CE distribution, but the ScaLAPACK routine for the LU decomposition request that the matrix is distributed using square blocks, so that it cannot be used with the distribution by blocks of columns used for the communication-efficient DWT. Therefore, the matrix had to be redistributed to a 2DBC distribution after computing the DWT (The redistribution was done using the ScaLAPACK redistribution subroutine **pdgemr2d**).

So, the difference between execution times for both distributions can be assigned entirely to the redistribution of the matrix form the CE (column) distribution to a 2DBC distribution. The results in that case can be seen in Tables 3 and 4.

Table 1. CPU time (sec.) for the algorithm with 1 wavelet level, Scalapack distribution

N	768	1152	1536	1920	2304	2688	3072	3456	3840	4224
1 Processor	0.15	0.36	0.68	1.14	1.76	2.45	3.41	4.76	6.19	7.93
2×2 Processors	0.13	0.29	0.40	0.62	0.80	1.07	1.40	1.80	2.29	2.76
3×3 Processors	0.08	0.14	0.20	0.29	0.42	0.52	0.69	0.93	1.17	1.42
4×4 Processors	0.05	0.10	0.12	0.18	0.25	0.32	0.43	0.56	0.72	0.85

Table 2. CPU time (sec.) for the algorithm with 2 wavelet level, Scalapack distribution

N	768	1152	1536	1920	2304	2688	3072	3456	3840	4224
1 Processor	0.13	0.26	0.47	0.76	1.13	1.65	1.96	2.80	3.37	4.22
2×2 Processors	0.12	0.23	0.30	0.47	0.64	0.78	0.98	1.17	1.37	1.63
3×3 Processors	0.07	0.12	0.14	0.21	0.29	0.39	0.43	0.61	0.66	0.79
4×4 Processors	0.05	0.08	0.09	0.13	0.17	0.23	0.27	0.35	0.40	0.46

Table 3. CPU time (sec.) for the algorithm with 1 wavelet level, Communication-efficient distribution

N	768	1152	1536	1920	2304	2688	3072	3456	3840	4224
1 Processor	0.15	0.36	0.68	1.14	1.76	2.45	3.41	4.76	6.19	7.93
2×2 Processors	0.22	0.49	0.70	1.08	1.50	2.03	2.62	3.30	4.14	4.98
3×3 Processors	0.25	0.38	0.58	0.87	1.13	1.47	1.92	2.44	3.07	3.66
4×4 Processors	0.26	0.40	0.52	0.77	0.98	1.33	1.66	2.13	2.64	3.15

Table 4. CPU time (sec.) for the algorithm with 2 wavelet level, Communication-efficient distribution

N	768	1152	1536	1920	2304	2688	3072	3456	3840	4224
1 Processor	0.13	0.26	0.47	0.76	1.13	1.65	1.96	2.80	3.37	4.22
2×2 Processors	0.14	0.27	0.38	0.59	0.81	1.04	1.31	1.54	1.83	2.21
3×3 Processors	0.18	0.25	0.31	0.42	0.56	0.70	0.85	1.07	1.20	1.43
4×4 Processors	0.27	0.34	0.41	0.52	0.53	0.60	0.69	0.86	0.99	1.15

It is quite clear (and obvious) that the times using directly the ScaLAPACK distribution are significantly better than the sequential times and the times obtained with the C-E distribution. Considering that the parallel computation of the DWT is faster with the CE distribution than with the 2DBC distribution, this means that the time needed to redistribute the matrix (from the CE distribution to the 2DBC distribution) is quite large. The efficiencies obtained are not very good in any case, due to the high communications requirements of the LU decomposition and the matrix-vector products. However, the efficiency improves when the number of processors and the size of the problem increases; in the case of the 2DBC distribution, the improvement is more pronounced.

4 Conclusions

In this paper we have considered the computation of the parallel DWT of a matrix, and specially to its application in numerical linear algebra applications.

We have shown that the 2DBC distribution is an interesting distribution for computing the parallel DWT, since it would allow the direct use of ScaLAPACK routines. As an example, the Wavelet-Schur preconditioner has been parallelized with good results with respect to the sequential versions, and with respect to the same algorithm using the C-E DWT proposed by Nielsen and Hegland.

Acknowledgement

This work has been supported by Spanish MCYT and FEDER under Grant TIC2003-08238-C02-02.

References

1. L.S. Blackford et al., Software, Enviroment, Tools. ScaLAPACK Users Guide, SIAM (1997).
2. T.F. Chan, K. Chen, On Two Variants of an Algebraic Wavelet Preconditioner, SIAM J. Sci. Comput.,24 (2002),260–283
3. O.M. Nielsen, M.A. Hegland, A Scalable Parallel 2D Wavelet Transform Algorithm. REPORT TR-CS-97-21, Australian National University, 1997. http://cs.anu.edu.au/techreports.
4. I. Daubechies, Ten Lectures on Wavelets, CBMS-NSF Regional Conference Series In Applied Mathematics, Vol. 61. Society for Industrial and Applied Mathematics Philadelphia, 1992.
5. M. V. Wickerhauser, Adapted Wavelet Analysis from Theory to software. A.K. Peters, IEEE Press, 1994.
6. G. Beylkin, R. Coifman, V. Rokhlin, Fast Wavelet Transforms and Numerical algorithms I, Commun. Pure Appl. Math., XLIV (1991), pp. 141-183.
7. T.F. Chan, W.P. Tang, W.L. Wan, Wavelet sparse approximate inverse preconditioners, BIT, 37 (1997), pp. 644-660.
8. R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, Society for Industrial and Applied Mathematics, Philadelphia, 1993.