PARALLEL IMPLEMENTATION OF THE YAU AND LU METHOD FOR EIGENVALUE COMPUTATION

Françoise Tisseur

DEPARTMENT OF COMPUTER SCIENCE,
UNIVERSITY OF TENNESSEE, KNOXVILLE, TN 37996-1301

Summary

In this paper, parallel extensions of a complete symmetric eigensolver, proposed by Yau and Lu in 1993, are presented. First, an overview of this invariant subspace decomposition method for dense symmetric matrices is given, followed by numerical results. Then, works are exposed in progress on distributed-memory implementation. The algorithm's heavy reliance on matrix-matrix multiplication, coupled with Fast Fourier Transform (FFT), should yield a highly parallelizable algorithm. Finally, performance results for the dominant computation kernel on the Intel Paragon are presented.

Address correspondence to Françoise Tisseur, Department of Computer Science, Room 111, Ayres Hall, University of Tennessee, Knoxville, TN 37996-1301, telephone (423) 974-8298, fax (423) 974-8296, e-mail ftisseur@cs.utk.edu.

The International Journal of Supercomputer Applications and High Performance Computing,
Volume 11, No. 3, Fall 1997, pp. 197-204
© 1997 Sage Publications, Inc.

1 Introduction

Recently, much effort has been devoted to develop efficiently traditional (Ipsen and Jessup, 1990; Bischof et al., 1994; Giménez et al., 1996) and new numerical algorithms (Cuppen, 1981; Dongarra and Sorensen, 1987) for the symmetric eigenvalue problem. In this paper, preliminary results on a new parallel algorithm for finding all the eigenvalues and eigenvectors of a dense symmetric matrix are presented. This algorithm is based on a recent method of Yau and Lu (1993). This method reduces the symmetric eigenvalue problem to a number of matrix multiplications. Yau and Lu's method involves approximating invariant subspaces of a special matrix using an FFT. The computation of the special matrix and the vectors for the FFT is rich in matrix-matrix multiplications. Matrix products can be implemented efficiently on most high performance machines and are often available as an optimized specific Level 3 BLAS library.

2 Yau and Lu Method

For computing invariant subspaces of a symmetric $n \times n$ matrix A with eigenvalues $\lambda_1, \ldots, \lambda_n$ and eigenvectors $x_1, \ldots, x_n$, Yau and Lu (1993) use a polynomial acceleration method. In the following, we give the basic idea.

For A such that $\mathrm{Sp}(A) \subset [0,\pi)$, we can consider the unitary matrix $B = e^{iA}$, whose eigenvalues all lie on the unit circle. Because A and B commute, they have the same eigenvectors.

$$\text{Let } P_N(z) = \sum_{j=0}^{N-1} \beta_j z^j \quad \text{be a polynomial of degree } N-1$$

that has a peak at $z = 1$ and is close to zero on the unit circle away from a vicinity of $z = 1$. Such a polynomial exists (Rivlin, 1990), and its coefficients β_j, $j = 0, \ldots, N-1$ can be obtained by a recursive formula (see Yau and Lu, 1993).

Starting from an initial vector, expanded in terms of the eigenvectors as $v_0 = \sum_{i=1}^{n} \alpha_i x_i$, we can define the function u: $[0, 2\pi) \to \mathbb{R}^n$ by

$$u(\lambda) = P_N(e^{-i\lambda}B)v_0 = \sum_{j=1}^{n} \alpha_j P_N(e^{i(\lambda_j - \lambda)})x_j.$$

If λ is chosen close to a particular λ_k and the other eigenvalues of B are not close to λ, then the coefficient of x_j will be small except when $j = k$. Thus, $u(\lambda)$ can be

viewed as an approximation of the eigenvector of B associated with the eigenvalue $e^{i\lambda}$.

Setting $v_j = B^j v_0$, the function $u(\lambda)$ can be written as

$$u(\lambda) = P_N(e^{-i\lambda}B)v_0 = \sum_{j=0}^{N-1} \beta_j B^j v_0 e^{-ij\lambda} = \sum_{j=0}^{N-1} \beta_j v_j e^{-ij\lambda},$$

where $\beta_j v_j$ are the Fourier coefficients of u. Therefore, the FFT can be used to compute $u(\lambda)$ at many different values of λ simultaneously. Then we need to select vectors $u(\lambda)$ that can be taken as eigenvectors, group them into p orthogonal clusters, and add more vectors if necessary. These p clusters form an orthogonal basis, $W = [W_1, \ldots, W_p]$, whose elements span invariant subspaces of A, and hence application of A to W decouples the spectrum:

$$W^T A W = \begin{pmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_p \end{pmatrix}$$

So, the initial problem is reduced to a small symmetric matrix eigenvalue problem in each cluster. Note that the subproblems $A_1, \ldots, A_p$ can be solved totally independently. The algorithm is presented in next section.

"The Yau and Lu method is a polynomial acceleration method especially written to introduce parallelizable computations."

"The symmetric eigenvalue problem is reduced to a number of matrix multiplications."

"The algorithm's heavy reliance on matrix-matrix multiplication, coupled with FFT, should yield a highly parallelizable algorithm."

3 Numerical Algorithm

Consider a real symmetric matrix A. The following steps find the eigenvalues and eigenvectors of A to the desired precision.

1. *Scaling and translation:* Compute upper and lower bounds of the spectrum of A and use these bounds to scale and translate the spectrum of A in $[0, 2\pi)$.
2. *Polynomial computation:* Let T_{N-1} be the Chebyshev polynomial of degree N – 1. The degree N is chosen such that

$$\frac{1}{T_{N-1}\left(\left(3 - \cos\frac{\pi}{n}\right) \Big/ \left(1 + \cos\frac{\pi}{n}\right)\right)} \le \kappa,$$

where κ is a measure of the desired accuracy of the computed invariant subspace.

Compute the coefficients $\beta_0^{(N-1)}, \ldots, \beta_{N-1}^{(N-1)}$ of the polynomial P_N by the recursive formulas:

$$\beta_0^{(0)} = 1, \ \beta_0^{(1)} = b, \ \beta_1^{(1)} = a$$

$$\beta_0^{(k+1)} = a\beta_1^{(k)} + 2b\beta_0^{(k)} - \beta_0^{(k-1)}$$

$$\beta_1^{(k+1)} = a\left(2\beta_0^{(k)} + \beta_2^{(k)}\right) + 2b\beta_1^{(k)} - \beta_1^{(k-1)}$$

$$\beta_j^{(k+1)} = a\left(\beta_{j-1}^{(k)} + \beta_{j+1}^{(k)}\right) + 2b\beta_j^{(k)} - \beta_j^{(k-1)} \text{ for } j > 1$$

$$\beta_j^{(k+1)} = 0 \text{ for } j > k+1,$$

where

$$a = \frac{2}{1+\cos(\pi/n)}, \quad b = \frac{1-\cos(\pi/n)}{1+\cos(\pi/n)}.$$

3. *Unitary matrix:* Compute matrices $C = \cos(\pi X)$ and $S = X^{-1}\sin(\pi X)$, where $X = \frac{A}{\pi} - I$ using the following Chebyshev expansion:

$$\cos(\pi x) \simeq c_0 + c_1 T_2(x) + c_2 T_4(x) + \cdots + c_5 T_{10}(x)$$
$$+T_{10}(c_6 T_2(x) + c_7 T_4(x) + \cdots + c_{10} T_{10}(x))$$

$$\frac{\sin(\pi x)}{x} \simeq s_0 + s_1 T_2(x) + s_2 T_4(x) + \cdots + s_5 T_{10}(x)$$
$$+T_{10}(s_6 T_2(x) + s_7 T_4(x) + \cdots + s_{10} T_{10}(x)),$$

where $c_0, \ldots, c_{10}$ and $s_0, \ldots, s_{10}$ are the Chebyshev coefficient given in Yau and Lu (1993).

4. *Computation of vectors* $v_j = e^{ijA}v_0$, $j = 0, 2N-1$: The real and imaginary part of v1 is obtained with the approximation C and S of cos (p X) and X^{-1} sin (p X):

$$v_1 = Cv_0 + S(Xv_0).$$

The remaining vectors can be computed following these $M = (\log_2 N - 1)$ steps:

Step 1: $C_1 = \cos(A)$

$$v_2 = 2C_1 v_1 - v_0$$

Step 2: $C_2 = 2C_1^2 - I$

$$(v_3, v_4) = 2C_2(v_1, v_2) - \overline{(v_1, v_0)}$$

Step 3: $C_3 = 2C_2^2 - I$

$$(v_5, v_6, v_7, v_8) = 2C_3(v_1, v_2, v_3, v_4) - \overline{(v_3, v_2, v_1, v_0)}$$

⋮ ⋮

Step M: $C_p = 2C_{p-1}^2 - I$

$$(v_{\frac{N}{2}-1}, \cdots, v_N) = 2C_p(v_1, \cdots, v_{\frac{N}{2}}) - \overline{(v_{\frac{N}{2}-1}, \cdots, v_0)}$$

5. *Evaluation of* $u(\lambda)$: Via the FFT, compute the vectors

$$u_k = u(\lambda)_{\lambda = \frac{k\pi}{N}} = \mathrm{Re}\sum_{j=0}^{N-1} \beta_j v_j e^{-j\frac{k\pi}{N}}, \quad \text{for } k = 0, \ldots, 2N-1.$$

6. *Selection and refinement:* Select the most useful vectors from the $2N$ vectors, $u_0, \ldots, v_{2N-1}$—that means vectors whose norm is not too small. Group them into a number of orthogonal clusters; add more vectors if necessary and reduce the initial problem to a small symmetric matrix eigenvalue problem in each cluster.

The most time-consuming part of the algorithm is the computation of the $2N$ vectors v_j. We need $(\log_2 N - 1)$ multiplications of two real symmetric matrices and $(\log_2 N - 1)$ more multiplications between a symmetric matrix and a rectangular one. If we consider an operation as one floating point computation, this part needs $(\log_2(N) - 1)n^3 + 4Nn^2$ operations.

The computation of e^{iA} requires six multiplications of real symmetric matrices for cos πX and one more for sin $\frac{\pi X}{X}$, that is, $7/2n^3$ operations.

The step of computing the u_k is still efficient because of the FFT algorithm and can be done in $nN \log_2(N)$ operations.

When necessary, the work for the supplementary vectors involves $8/3n^3$ more operations because we need to construct an orthogonal matrix by a QR factorization. The reduction $W^T AW$ involves two matrix multiplications and is done in $3/2n^3$ operations. Usually, the subproblems in each cluster only involve small matrices, and the cost is negligible compared to the total work.

So, the total amount of operations is given by:

$$\text{Total operation} = \left(\frac{16}{3} + \log_2(N)\right)n^3 + 4Nn^2 + O(n^2).$$

The sequential complexity of the Yau and Lu algorithm is considerably greater than that of other algorithms. Meanwhile, note that Steps 3, 4, and 6 of the algorithm are all based on matrix-matrix multiplications, and if we suppose that $N \simeq 8n$, then

$$\frac{\text{Total operations involving matrix multiplications}}{\text{Total operations}} \geq$$

$$\frac{35 + \log_2(N)}{38 + \log_2(N)}.$$

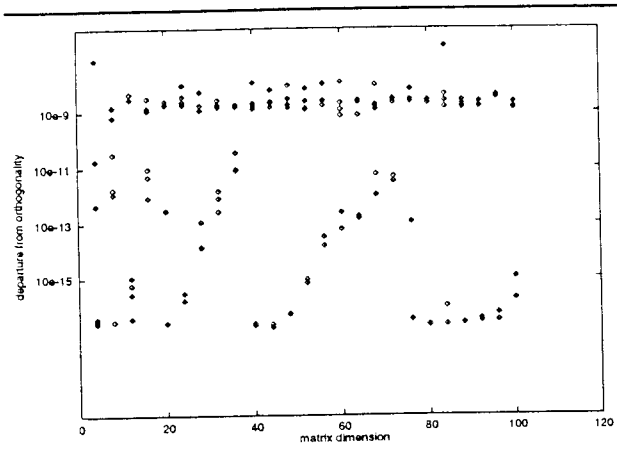

Fig. 1 Departure from orthogonality for dense symmetric matrices

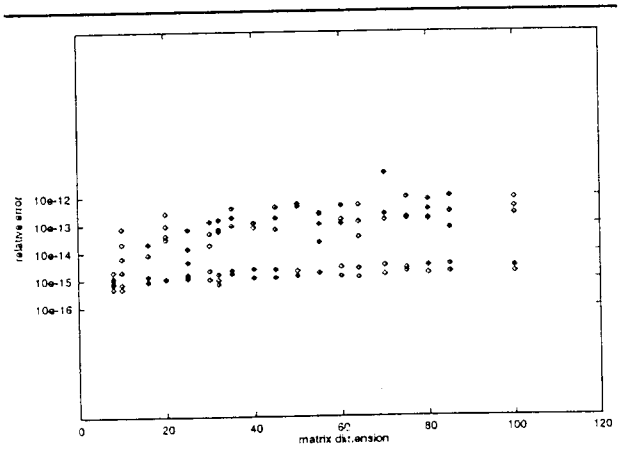

Fig. 2 Relative error on computed eigenvalues

So, for matrices of a dimension between 500 and 1,000, we find that matrix multiplications are more than 90% of the total operations count. For larger dimensions of the matrix A, this percentage will of course increase. On parallel computers, the efficiency of BLAS 3 routines can justify the use of the extra multiplications.

4 Numerical Results

We have tested our algorithm on a large set of test matrices using the LAPACK (Anderson et al., 1995) test generation routine DLATMS. This routine constructs symmetric matrices of the form

$$A = U^T D U,$$

where U is a random orthogonal matrix and $D = \text{diag}\,(\lambda_1, \ldots, \lambda_n)$ a diagonal matrix.

The accuracy of eigensubspaces is controlled in Step 2 of the algorithm. Because we have chosen N such that $\kappa < 10^{-9}$, we expect nine accurate digits for the eigenvectors. To verify this, we have computed (see Figure 1) the departure from orthogonality, given by

$$\max_{ij} \left| (Q^T Q - I_n)_{ij} \right|,$$

where Q is the matrix of eigenvectors.

Accuracy in the computed eigenvalues can be quantified by computing the relative error

$$\max_{1 \le i \le n} \frac{|\lambda_i - \hat{\lambda}_i|}{|\lambda_i|},$$

where λ_i denotes an exact eigenvalue and $\hat{\lambda}_i$ the corresponding computed eigenvalue (see Figure 2).

Accuracy in the residuals for a given matrix A is quantified by computing the maximum normalized two-norm residual

$$\max_i \frac{\|A\hat{x}_i - \hat{\lambda}_i \hat{x}_i\|_2}{\|A\|_F}, \qquad \text{with } \|\hat{x}_i\|_2 = 1,$$

where $\hat{x}_i$ is the computed eigenvector corresponding to the computed eigenvalue $\hat{\lambda}_i$ (see Figure 3).

All the test results presented in this section where performed on a SUNsparc 512 MP. The arithmetic was IEEE standard double precision, with a machine precision of $\varepsilon = 2^{-53} \approx 2.22044 \times 10^{-16}$ and an over- or underflow threshold of $10^{\pm 307}$.

5 Parallel Implementation

There are two forms of parallelism in the Yau and Lu (1993) method. The first one corresponds to data parallelism with the heavy reliance on matrix-matrix multiplications. The second one is a kind of functional parallelism with the reduction of the initial problem to a number of small symmetric eigenvalue problems that can be solved totally independently on each processor. That is why we say that the Yau and Lu method yields a highly parallelizable algorithm.

In Section 3, we have shown that the computational cost of the algorithm is dominated by dense matrix-matrix multiplications. So, the performance of this algorithm will depend heavily on the matrix multiplication code. We need two different types of matrix-matrix products. The first one is the product between symmetric and rectangular matrices, and the second one is the product between two symmetric matrices that commute. Therefore, this last product is also symmetric. This should be exploited in implementing a specific distributed matrix multiplication algorithm for commutative symmetric matrices. Such an algorithm can be designed using some techniques presented by Snyder (Lin and Snyder, 1992). However, up to now, our code uses the standard parallel BLAS 3 **pdgemm** available in the PBLAS of ScaLAPACK (Choi et al., 1994). For the moment, many distributed-memory machines do not overlap communications and computations. For this reason and for the portability of our algorithm, we use the Basic Linear Algebra Communication Subprograms (BLACS) for the communications. Hence, our code is highly synchronous.

The upper and lower bounds of the spectrum of A are estimated by using a method developed by Rojo and Soto (1994). From point-of-view computation, this method needs matrix-matrix products. It does not increase the total computational part because we can reuse results for the construction of Chebyshev polynomials. We focus our parallelization on the most cost-effective part of the algorithm, that is, the construction of the matrices C, S, and the Fourier's coefficients. Figures 4 and 5 present the parallel algorithms.

For parts 1 through 4 of our algorithm, we use a block-cyclic storage of the matrix (Figure 6). But to do the FFT in sequential order on each processor, we prefer to redistribute the matrix of $2N$ complexes vectors v_j by processor rows with the routine PDGMR2D of ScaLAPACK (Choi et al., 1994). This storage is also useful for the selection.

For the last part of the algorithm, each processor solves its own small symmetric eigenvalue problem using

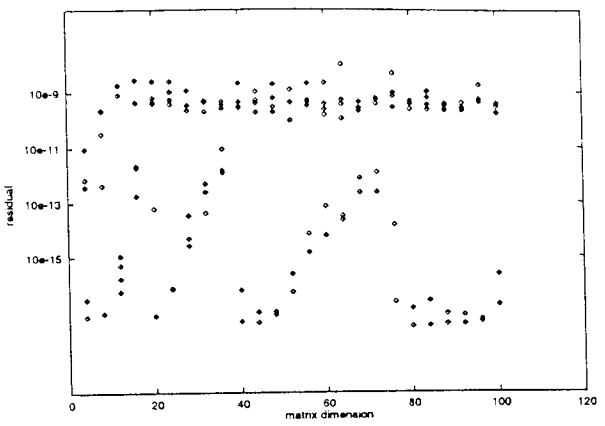

Fig. 3 Residuals for dense symmetric matrices

```
/*Computation of C and S*/
/*Construction of T1,T2,T4,T8,T10.*/
T2:=2AA-I   /* Parallel BLAS 3 */
FOR i=4 to 10 step 2
      Ti:=2Ti-2T2-Ti-4   /* Parallel BLAS 3 */
ENDFOR
/*Chebyshev expansion of C and S*/
C:=c0I, S:=s0I
FOR i=1 to 5
      C:=ciT2i+C
      S:=siT2i+S
      CC:=ci+5T2i+CC
      SS:=si+5T2i+SS
ENDFOR
C:=-T10CC+C   /* Parallel BLAS 3 */
S:=-T10SS+S   /* Parallel BLAS 3 */
```

Fig. 4 Algorithm for computing C and S

/* Initialization */
$v_1 = Cv_0 + iSv_0$
$v_2 = 2Cv_1 - v_0$

/* Computing first Q vectors */
FOR $i = 1$ to $\log_2(Q)$ do
  $C := 2C^2 - I$
  $(v'_{2^i+1}, v'_{2^i+2}, \ldots, v'_{2^{i+1}+1}) := 2C(v_1, v_2, \ldots, v_{2^i})$
  IF MYCOL = 0 THEN
    $(v_{2^i+1}, v_{2^i+2}, \ldots, v_{2^{i+1}+1}) := (v'_{2^i+1}, v'_{2^i+2}, \ldots, v'_{2^{i+1}+1}) - \overline{(v_{2^i-1}, v_{2^i-2}, \ldots, v_0)}$
  ENDIF
ENDFOR

/* Distribution of first Q vectors */
IF MYCOL = 0 THEN
    FOR $i = 1$ to $Q - 1$
      send vector v_{i+1} to column i
    ENDFOR
ELSE
  receive vector v_1 from column 0
ENDIF

/* Main loop */
FOR $i = 1$ to $\log_2(\frac{N}{2})$ do
  $C := 2C^2 - I$
  $ind := 2^{(i-1)} * \sqrt{p} + ncol + 1;$
  $(v'_{ind}, v'_{ind+\sqrt{p}}, \ldots, v'_{ind+2^{(i-1)}\sqrt{p}}) := 2C(v_{ind-2^{(i-1)}\sqrt{p}}, \ldots, v_{ind-2\sqrt{p}}, v_{ind-\sqrt{p}})$
  Exchange Data
  $[v_{ind}, v_{ind+\sqrt{p}}, \ldots, v_{ind+2^{(i-1)}\sqrt{p}}) := (v'_{ind}, v'_{ind+\sqrt{p}}, \ldots, v'_{ind+2^{(i-1)}\sqrt{p}}]$
  $\quad - \overline{(v_{(2^{(i-1)}\sqrt{p}-(ncol+1)}, \ldots, v_{2\sqrt{p}-(ncol+1)}, v_{\sqrt{p}-(ncol+1)})}$
ENDFOR

Fig. 5 Algortihm for computing v_j for $j = 0, \ldots, N-1$

DSYEV (QR algorithm) from LAPACK (Anderson et al., 1992).

Preliminary results on the Paragon have been obtained (see Figures 7 and 8). We have good speedup and efficiency. Due to memory limitation and cache size, we get speedup up to 16 and superlinearity for 4×4 grids.

6 Conclusion

We studied a new approach to compute the eigenvalues and eigenvectors of a real symmetric matrix. The algo-

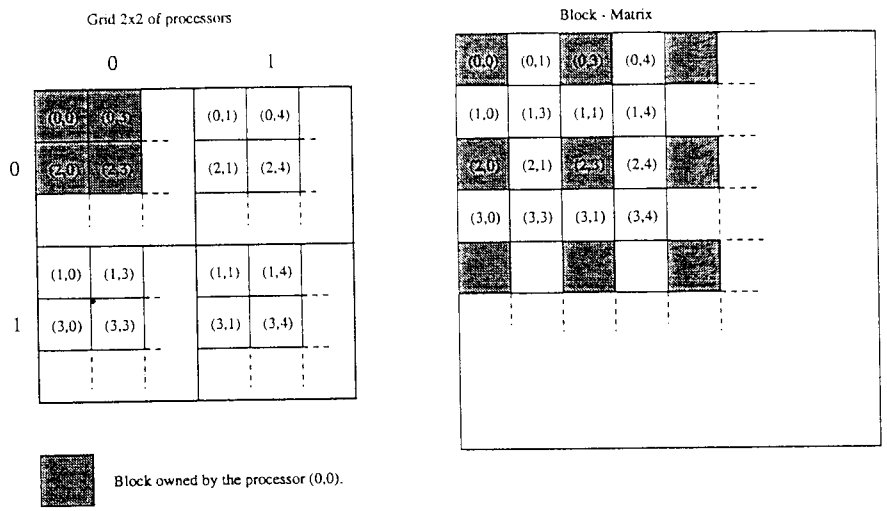

Fig. 6 Block cyclic distribution on a 2 × 2 processor grid

rithm parallelized in this paper is not efficient on a sequential machine but can fully take advantage of parallel machines because of less data decencies and the use of matrix products as the most important computed kernel. We obtain good performances for our code concerning efficiency and execution time. As we have not yet implemented our optimized matrix-matrix product or focused our work on the last part of the algorithm, we expect our results to improve.

BIOGRAPHY

Françoise Tisseur is a postdoctoral research associate in the Department of Computer Science at the University of Tennessee. She received her Ph.D. in numerical analysis from Saint-Etienne University, France, in 1997. Her primary research interests are in dense linear algebra, eigensolvers, and parallel scientific computing.

REFERENCES

Anderson, E., Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. DuCroz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. 1992. *LAPACK user's guide*. Philadelphia: Society for Industrial and Applied Mathematics.

Anderson, E., Z. Bai, C. H. Bischof, J. W. Demmel, J. J. Dongarra, J. J. DuCroz, A. Greenbaum, S. J. Hammarling, A. McKenney, S. Ostrouchov, and D. C. Sorensen. 1995. *LAPACK users' guide, release 2.0*. 2d ed. Philadelphia: SIAM.

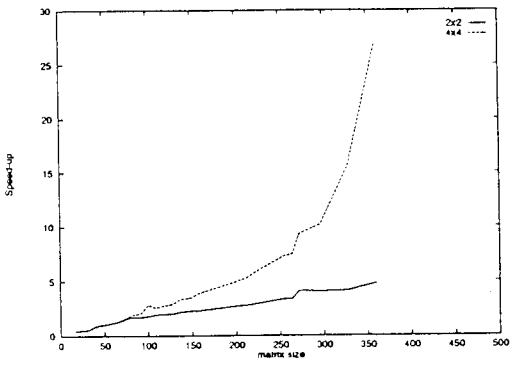

Fig. 7 Speedup for a 2 × 2 and 4 × 4x grid according to the size of the matrix on Paragon

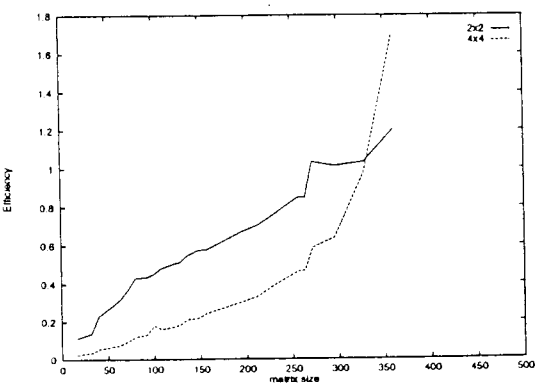

Fig. 8 Efficiency for a 2 × 2 and 4 × 4 grid according to the size of the matrix on Paragon

Bischof, C., S. Huss-Lederman, X. Sun, A. Tsao, and T. Turnbull. 1994. Parallel performance of a symmetric eigensolver based on the invariant subspace decomposition approach. PRISM Working Note 15, Argonne National Laboratory, Argonne, IL.

Choi, J., J. J. Dongarra, D. W. Walker, and R. C. Whaley. 1994. Scalapack reference manual parallel factorization routines (LU, QR, and Cholesky) and parallel reduction routines (HDR, BRD, and TRD). Technical Report ORNL/TM-12470, Oak Ridge National Laboratory, Oak Ridge, TN.

Cuppen, J.J.M. 1981. A divide and conquer method for the symmetric tridiagonal eigenproblem. *Numer. Math.* 36: 177-195.

Dongarra, J., and D. Sorensen. 1987. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Sci. Statist. Comput.* 8:S139-S154.

Giménez, D., R. van de Geijn, V. Hernández, and A. M. Vidal. 1996. Exploiting the symmetry on the Jacobi method on a mesh of processors. Paper presented at the 4th EUROMICRO Workshop on Parallel and Distributed Processing, Braga, Portugal.

Ipsen, I.C.F., and E. R. Jessup. 1990. Solving the symmetric tridiagonal eigenvalue problem on the hypercube. *SIAM J. Sci. Stat. Comput.* 11 (2):203-229.

Lin, C., and L. Snyder. 1992. A matrix product algorithm and its comparative performance on hypercubes. In *Scalable High Performance Computing Conference SHPCC92.* Los Alamitos, CA: IEEE Computer Society, pp. 190-193.

Rivlin, T. J. 1990. *Chebyshev polynomial: From approximation theory to algebraic and number theory.* 2d ed. New York: John Wiley.

Rojo, O., and R. L. Soto. 1994. A decreasing sequence of eigenvalue localization region. *Linear Alg. Appl.* 196:71-84.

Yau, S.-T., and Lu, Y. 1993. Reducing the symmetric matrix eigenvalue problem to matrix multiplications. *SIAM Journal Sci. Comput.* 14 (1):121-136.