Open access • Journal Article • DOI:10.1016/0167-8191(92)90059-G

Divide and conquer algorithms for the bandsymmetric eigenvalue problem

Peter Arbenz

Published on: 01 Oct 1992 - Parallel Computing

Topics: Divide-and-conquer eigenvalue algorithm, Akra–Bazzi method, Divide and conquer algorithms, Tridiagonal matrix and QR algorithm

Divide and conquer algorithms for the bandsymmetric eigenvalue problem

Eidgenössische
Technische Hochschule
Zürich

Departement Informatik
Institut für
Wissenschaftliches Rechnen

Peter Arbenz

Divide and Conquer Algorithms for the Bandsymmetric Eigenvalue Problem

November 1991

170

Authors´ address:

Institut für Wissenschaftliches Rechnen
ETH Zentrum
CH-8092 Zurich, Switzerland.
e-mail: arbenz@inf.ethz.ch

Divide and Conquer Algorithms for the Bandsymmetric Eigenvalue Problem

Peter Arbenz

ETH Zentrum, Institut für Wissenschaftliches Rechnen, 8092 Zürich, Switzerland
(e-mail: arbenz@inf.ethz.ch)

Abstract.
Divide and conquer algorithms are formulated for the solution of the eigenvalue problem for symmetric band matrices. The new algorithms are compared to the traditional solution paths offered by EISPACK, tridiagonalization of the bandmatrix followed by the tridiagonal QR algorithm.

Keywords. eigenvalue problem, symmetric band matrix, divide and conquer algorithm

1. Introduction

In this paper we deal with the eigenvalue problem

$$A\mathbf{x} = \lambda\mathbf{x}, \tag{1.1a}$$

where

$$A = \begin{pmatrix} a_{00} & \cdots & a_{0r} & & & & \\ \vdots & \ddots & & \ddots & & & \\ a_{r0} & & \ddots & & \ddots & & \\ & \ddots & & \ddots & & a_{n-r-1,n-1} & \\ & & \ddots & & \ddots & \vdots & \\ & & & a_{n-1,n-r-1} & \cdots & a_{n-1,n-1} & \end{pmatrix} \in \mathbf{R}^{n\times n} \tag{1.1b}$$

is a symmetric band matrix with semi-bandwidth r, i.e. the elements a_{ij} vanish if $|i-j| > r$. Eigenvalue problems of this type are, for example, obtained when one-dimensional vibrating systems are approximated by finite elements or splines. A sequence of increasingly larger bandsymmetric eigenvalue problems is generated in the course of the *block* Lanczos algorithm [28, p.285].

The 'classical' approach to this problem is to first transform A into a similar tridiagonal matrix T and then solve the eigenvalue problem for T by the QR or QL algorithm [34, Contributions II/2-4 and II/8] (See also [20, §8.2] or [28, §8].) The subroutines in EISPACK [31],[18] for this purpose are `tred2`, `bandr` for the reduction to tridiagonal form and `tql2` for QL algorithm. (The subroutine `bandr` however should not be used unless only the eigenvalues without eigenvectors are

to be computed.) An alternative to these solution paths is to apply the QR algorithm directly to the band matrix [34, Contribution II/7]. This solution path — recommended if only a few of the eigenvalues are desired — is incorporated in the EISPACK subroutine bqr.

With the advent of parallel computers new algorithms are being investigated to permit a better utilization of the new hard and software. A prominent example of an algorithm of this kind is the divide and conquer algorithm introduced by Cuppen for the solution of the symmetric tridiagonal eigenvalue problem [12]. This algorithm has been improved and implemented by Dongarra and Sorensen [14] on shared memory machines. They report on very favorable results. In addition, Ipsen and Jessup [21] implemented Cuppen's algorithm on a hypercube, a parallel computer with distributed memory. They did not obtain as good results due to the large amount of data transfer required by the divide and conquer algorithm.

The symmetric tridiagonal eigenvalue problem has also been solved by Lo, Philippe and Sameh [24] by an algorithm based on bisection/multisection and inverse vector iteration. The bisection method used for extracting the eigenvalues is accelerated by the rootfinder zeroin applied on $\det(A-\lambda)$. A similar approach has been taken by Ma, Patrick and Szyld [25] for the banded generalized eigenvalue problem. These approaches are better suited than the divide and conquer algorithms when only a few of the eigenvalue and eigenvectors are to be computed.

In this paper we examine divide and conquer algorithms for solving (1.1). The special case of this problem, where A a banded symmetric *Toeplitz* matrix was recently investigated by the author [2]. In this paper we consider the general case. We comment on the banded generalized eigenvalue problem in section 5.

A divide and conquer algorithm consists of essentially three parts. First the problem at hand is divided in several in some sense simpler problems which are more easily managable. In a second part these simpler problems are solved. The most difficult third part consists of reassembling the original problem by making use of the solutions of the already solved simple problems. These three parts will be dealt with in sections 2 to 4. In §§6 and 7 we will formulate an algorithm for the band matrix eigenvalue problem and discuss some numerical experiments. In §8 we deal with a variant of the algorithm of §6.

Throughout this paper uppercase, boldface and lowercase letters will denote matrices, vectors and scalars, respectively.

2. The divide part

We can treat (1.1) by formulating either a *modified*, a *restricted*, or an *extended* eigenvalue problem. We deal with these approaches in this section.

2.1. Divide and conquer by modification

A modified eigenvalue problem is formed by splitting A as follows:

$$A = A_0 + D := \begin{pmatrix} A_1 & O \\ O & A_2 \end{pmatrix} + D, \qquad A_i \in \mathbf{R}^{k_i \times k_i}, \quad k_1 + k_2 = n, \tag{2.1}$$

where A_1 and A_2 are again band matrices with semi-bandwidth r. We will investigate several ways to define the matrix D. As the complexity of the computation of the spectral decomposition of A from that of A_0 in the reassembly part is proportional to the square of the rank of D (cf. §4) it is advantageous to keep this number as low as possible. Dongarra and Sorensen [14] proposed to choose

$$D = \sum_{i>k_1, j\le k_1} a_{ij}(\vartheta_{ij}\mathbf{e}_j + \mathbf{e}_i)(\mathbf{e}_j + \vartheta_{ij}\mathbf{e}_i)^T, \qquad \vartheta_{ij} = \pm 1, \tag{2.2}$$

and thus perform $r(r+1)/2$ consecutive rank-one updates, an approach which is directly inspired by Cuppen's algorithm. Here, $\mathbf{e}_i$ denotes the i-th axis vector. Note that although D is defined as the sum of $r(r+1)/2$ rank-1 matrices, its rank can never exceed $2r$. If we choose $\vartheta_{ij} = -1$ for all i, j, then the rank of D would be at most $2r-1$ as $D\mathbf{e} = \mathbf{0}$ where $\mathbf{e} = \sum_{i=k_1-r}^{k-1+r} \mathbf{e}_i$. A similar argument holds if we set $\vartheta_{ij} = +1$ for all i, j.

In their divide and conquer algorithm for the tridiagonal eigenvalue problem Dongarra and Sorensen [14] chose $\vartheta_1 = 1$ and $\delta = \delta_1 = \pm a_{k_1+1,k_1}$. They determined the sign of δ such that cancellation is minimized when constructing A_1 and A_2. This strategy proved satisfactory in practical experiments.

It is possible to have rank $D = r$ if one defines for instance

$$D = \sum_{j=1}^{r} \delta_j \mathbf{v}_j \mathbf{v}_j^T, \qquad \delta_j \neq 0, \tag{2.3a}$$

where

$$\mathbf{v}_j := \vartheta_j \mathbf{e}_{k_1-r+j} + \frac{1}{\delta_j \vartheta_j} \sum_{\ell=1}^{j} a_{k_1+\ell,k_1-r+j}\mathbf{e}_{k_1+\ell}, \qquad \vartheta_j \neq 0.$$

In matrix notation this is

$$D = V\Delta V^T, \qquad V = [\mathbf{v}_1, \ldots, \mathbf{v}_r] = \begin{pmatrix} O \\ \Theta \\ V_1\Theta^{-T}\Delta^{-1} \\ O \end{pmatrix} \in \mathbf{R}^{n\times r}, \tag{2.3b}$$

where $\Theta, \Delta \in \mathbf{R}^{r\times r}$ are diagonal matrices and where

$$V_1 = \begin{pmatrix} a_{k_1+1,k_1-r+1} & \cdots & a_{k_1+1,k_1} \\ & \ddots & \vdots \\ & & a_{k_1+r,k_1} \end{pmatrix} \in \mathbf{R}^{r\times r}$$

is the part of A which is to be 'cut out'. More generally we may admit any regular matrices Θ and Δ, $\Delta = \Delta^T$. Introducing $M := \Theta\Delta\Theta^T$ we can write (2.3b) as

$$D = \begin{pmatrix} M \\ V_1 \end{pmatrix} M^{-1} \begin{pmatrix} M \\ V_1 \end{pmatrix}^T = \begin{pmatrix} M & V_1^T \\ V_1 & V_1 M^{-1} V_1^T \end{pmatrix} \tag{2.4}$$

Evidently it is not possible in general to construct a modification D with rank lower than r.

In the case where $r > 1$ it is not clear how to choose the modification D reasonably. If we set $M = I$, D is in most cases unbalanced, meaning that $\|V_1^T M^{-1} V_1\|$ can be much smaller or larger than $\|M\| = 1$. When choosing $M := -(V_1^T V_1)^{1/2}$ we get a matrix which is balanced in that sense. If $V_1 = X\Sigma Y^T$ is the singular value decomposition of V_1 then $M = -Y\Sigma Y^T$ and $V_1 M^{-1} V_1^T = -X\Sigma X^T$. So both these matrices are symmetric negative definite and have the same norm and condition number as V_1. The modification D obtained this way has the form (2.3) with $\Theta = Y$, $\Delta = -\Sigma = \mathrm{diag}(-\sigma_1, \ldots, -\sigma_r)$. If we assume that A is positive definite then positive definite matrices are added together when $A_0 = A - D$ is formed. In this positive definite case we would get the same matrices as Dongarra and Sorensen for $r = 1$. The above definition of M of course does not exclude cancellations on the element level, an issue which could be treated more easily by an approach like the one in (2.2) but to the price of an unnecessarily high rank for D. Computing the singular value decomposition of V_1 costs $O(r^3)$ floating point operations. Assuming that $r \ll n$ and that only very few decompositions are needed, the computation cost for the SVD is negligible compared to the overall cost of the entire algorithm. Furthermore, the SVD yields the actual rank of V_1 and makes it possible to make advantage of eventual rank deficiencies.

There is a close relationship between the eigenvalues of A and A_0. Let $\lambda_0 \le \cdots \le \lambda_{n-1}$ and $\lambda_0^{(0)} \le \cdots \le \lambda_{n-1}^{(0)}$ denote these eigenvalues. If Δ (or M) has r_- negative and $r_+ = r - r_-$ positive eigenvalues the *interlacing properties*

$$\lambda_{j-r_-}^{(0)} \le \lambda_j \le \lambda_{j+r_+}^{(0)}, \qquad 0 \le j < n, \tag{2.5}$$

hold if we set $\lambda_j^{(0)} = -\infty$ for $j < 0$ and $\lambda_j^{(0)} = +\infty$ for $j \ge n$ [4]. These inequalities mean that the eigenvalues of A and of A_0 are distributed in similar way. In cases where (2.5) doesn't yield finite bounds we can use the inequalities

$$\lambda_0^{(0)} - \|\Delta\|_2 \|V\|_2^2 \le \lambda_j \le \lambda_{n-1}^{(0)} + \|\Delta\|_2 \|V\|_2^2, \qquad 0 \le j < n, \tag{2.6}$$

or Gerschgorin's circle theorem [20]. Because $\|V\|_2^2 \le \|V\|_F^2 = \sum_{i,j} v_{i,j}^2$, eigenvalue bounds for λ_j are easy and cheap to compute. However, the bounds obtained by (2.6) are crude in most cases.

2.2. Divide and conquer by restriction

Instead of adding a modification to the original matrix A to obtain a reducible matrix A_0 as in (2.1), it is possible to construct a matrix (again called) A_0 of order greater than n which contains A in some sense. One can imagine that A is stretched until it is teared in two pieces, whereby elements around the tearing point contribute to both pieces. To be specific, let us define

$$A_0 := A_1 \oplus A_2, \qquad A_i \in \mathbf{R}^{k_i \times k_i}, \quad k_1 + k_2 = n + r, \tag{2.7}$$

with

$$A_i := \begin{pmatrix} a_{00}^{(i)} & \cdots & a_{0r}^{(i)} & & & & \\ \vdots & \ddots & & \ddots & & & \\ a_{r0}^{(i)} & & \ddots & & \ddots & & \\ & \ddots & & \ddots & & a_{k_i-r-1,k_i-1}^{(i)} & \\ & & \ddots & & \ddots & \vdots & \\ & & & a_{k_i-1,k_i-r-1}^{(i)} & \cdots & a_{k_i-1,k_i-1}^{(i)} & \end{pmatrix}, \qquad i = 1,2,$$

where

$$a_{ij}^{(1)} = \begin{cases} a_{ij}, & i < k_1 - r \text{ or } j < k_1 - r, \\ \frac{1}{2}a_{ij}, & k_1 - r \le i,j < k_1, \end{cases}$$

and

$$a_{ij}^{(2)} = \begin{cases} \frac{1}{2}a_{k_1-r+i,k_1-r+j}, & 0 \le i,j < r, \\ a_{k_1-r+i,k_1-r+j}, & i \ge r \text{ or } j \ge r, \end{cases}$$

respectively. So the $r \times r$ block

$$\begin{pmatrix} a_{k_1-r,k_1-r} & \cdots & a_{k_1-r,k_1-1} \\ \vdots & & \vdots \\ a_{k_1-1,k_1-r} & \cdots & a_{k_1-1,k_1-1} \end{pmatrix}$$

appears in A_1 as well as in A_2 with weight $\frac{1}{2}$.

In addition, we define the mapping

$$\begin{aligned} ' : (\mathbf{R}^n, \|.\|_2) \quad & \ni \mathbf{x} = (x_0, \ldots, x_{n-1})^T \\ & \longmapsto \mathbf{x}' := (x_0, \ldots, x_{k_1-1}, x_{k_1-r}, \ldots, x_{n-1})^T \in (\mathbf{R}^{n+r}, |||.|||) \end{aligned} \tag{2.8}$$

where

$$|||\mathbf{y}||| := \left(\sum_{i=0}^{k_1-r-1} y_i^2 + \frac{1}{2} \sum_{i=k_1-r}^{k_1+r-1} y_i^2 + \sum_{i=k_1+r}^{n+r-1} y_i^2 \right)^{1/2}, \qquad \mathbf{y} \in \mathbf{R}^{n+r}.$$

Notice that the map $'$ duplicates the elements x_{k_1-r} to x_{k_1-1}. $'$ is an isometry from $(\mathbf{R}^n, \|.\|_2)$ onto the subspace of those vectors $\mathbf{y}$ in $(\mathbf{R}^{n+r}, |||.|||)$ which satify

$$y_{k_1-r+i} = y_{k_1+i}, \qquad 0 \le i < r,$$

i.e., which are orthogonal to the column space $\mathcal{R}(V)$ of the matrix

$$V = \begin{pmatrix} O_{k_1-r,r} \\ I_r \\ -I_r \\ O_{k_2-r,r} \end{pmatrix} \in \mathbf{R}^{(n+r)\times r}.$$

We now consider the problem of finding the stationary values of the Rayleigh quotient

$$R[\mathbf{y}] = \frac{\mathbf{y}^T A_0 \mathbf{y}}{\mathbf{y}^T \mathbf{y}}, \qquad \mathbf{y} \in \mathcal{R}(V)^\perp.$$

Because of the *restriction* $V^T\mathbf{y} = \mathbf{0}$ the stationary values of $R[\mathbf{y}]$ are in general *not* the eigenvalues of A_0. Using Lagrange multipliers one sees that the problem is equivalent with finding values λ for which the 'restricted eigenvalue problem'

$$\begin{pmatrix} \lambda I - A_0 & V \\ V^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{y} \\ \mathbf{a} \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{2.9}$$

has nontrivial solutions [3]. These values are precisely the stationary values. The $\mathbf{y}$ component of a corresponding solution of (2.9) is a corresponding stationary point. From (2.9) we see that $\mathbf{y}$ is a stationary point of the restricted Rayleigh quotient if and only if

$$(A_0 - \lambda I)\mathbf{y} \in \mathcal{R}(V) \quad \text{and} \quad V^T\mathbf{y} = \mathbf{0}. \tag{2.10}$$

Let now $(\lambda, \mathbf{x})$ be an eigenpair of A in (1.1). Then by constuction $V^T\mathbf{x}' = \mathbf{0}$. If we form $\mathbf{y} := (A_0 - \lambda I)\mathbf{x}'$ then evidentially $y_i = 0$ for $i < k_1 - r$ and $i \geq k_1 + r$. Furthermore, $y_{k_1-r+j} + y_{k_1+j} = 0$ for $0 \leq j < r$. Thus, (2.10) holds. Therefore, the stationary values of the restricted Rayleigh quotient are the eigenvalues of A and the connection between A's eigenvectors and the stationary vectors is given by $'$.

In the search for stationary values of $R[\mathbf{y}]$, we will make use of the following bounds. The eigenvalues λ_j of A *interlace* the eigenvalues $\lambda_j^{(0)}$ of A_0. If we assume both eigenvalue sequences $\{\lambda_j\}_{j=0}^{n-r}$ and $\{\lambda_j^{(0)}\}_{j=0}^{n+r-1}$ to be ordered increasingly, the interlacing properties

$$\lambda_j^{(0)} \leq \lambda_j \leq \lambda_{j+r}^{(0)}, \qquad 0 \leq j < n, \tag{2.11}$$

hold.

Remarks. (1) In [1] a mapping similar to $'$ in (2.8) has been defined by

$$'' : \mathbf{x} \longmapsto (x_0, \ldots, x_{k_1-r-2}, \frac{x_{k_1-r-1}}{\sqrt{2}}, \ldots, \frac{x_{k_1-1}}{\sqrt{2}}, \frac{x_{k_1-r-1}}{\sqrt{2}}, \ldots, \frac{x_{k_1-1}}{\sqrt{2}}, x_{k_1}, \ldots, x_{n-1})^T$$

yielding an isometry from $\mathbf{R}^n$ onto $\mathcal{R}(V)^\perp \subset \mathbf{R}^{n+r}$, both spaces equipped with the Euklidian vector norm. Replacing $'$ by $''$ would cause changes in the matrix A_0.
(2) One could consider the approach of this section as a domain decomposition technique: Variables corresponding to the interfaces of two subdomains belong to either of the subdomains but are weighed only by a half. □

2.3. Divide and conquer by extension

Divide and conquer by extension is the reverse of divide and conquer by restriction. Here, a matrix A_0 is constructed such that the original matrix A contains A_0 as a submatrix. Therefore, A is called an extension of A_0. In this subsection we choose k_1 and k_2 such that $k_1 + k_2 + r = n$ and write

$$A := \begin{pmatrix} A_1 & V_1 & O \\ V_1^T & \Delta & V_2^T \\ O & V_2 & A_2 \end{pmatrix}, \qquad A_i \in \mathbf{R}^{k_i \times k_i}, V_i \in \mathbf{R}^{k_i \times r}, \Delta \in \mathbf{R}^{r \times r}. \tag{2.12}$$

Clearly, $A_0\mathbf{x} = \lambda\mathbf{x}$, $A_0 := A_1 \oplus A_2$, is equivalent with the restricted eigenvalue problem

$$\begin{pmatrix} \lambda - A & R \\ R^T & O \end{pmatrix}, \qquad R = \begin{pmatrix} O_{k_1 \times r} \\ I_r \\ O_{k_2 \times r} \end{pmatrix}. \tag{2.13}$$

So, we may consider $A\mathbf{y} = \lambda\mathbf{y}$ as an *extension* of $A_0\mathbf{x} = \lambda\mathbf{x}$. For the investigation in the following section 4 we apply on A in (2.12) a similarity transformation with the permutation matrix

$$P := \begin{pmatrix} I_{k_1} & O & O \\ O & O & I_r \\ O & I_{k_2} & O \end{pmatrix} \in \mathbb{R}^{n \times n}$$

to obtain

$$\bar{A} := P^T A P = \begin{pmatrix} A_0 & V \\ V^T & \Delta \end{pmatrix}, \qquad V = \begin{pmatrix} V_1 \\ V_2 \end{pmatrix}. \tag{2.14}$$

Considering A_0 as a restriction af A or $\bar{A}$, respectively, we get from (2.11) the interlacing properties

$$\lambda_{j-r}^{(0)} \le \lambda_j \le \lambda_j^{(0)}, \qquad 0 \le j < n, \tag{2.15}$$

where the $\lambda_j^{(0)}$ are the increasingly ordered $n - r$ eigenvalues of A_0. Here, we set $\lambda_j^{(0)} = -\infty$ for $j < 0$ and $\lambda_j^{(0)} = +\infty$ for $j \ge n - r$. Finite bounds for the extreme eigenvalues of A can be obtained by Gerschgorin's circle theorem or by $|\lambda_j| \le \|A\|$.

The technique of extended eigenvalue problems was used by O'Leary and Stewart [27] for computing the eigenvalues of arrow matrices and by Jessup [22] for solving the nonsymmetric tridiagonal eigenvalue problem in a divide and conquer approach.

2.4. Comparison of the approaches

The approach by modified eigenvalue problems has the disadvantage that it is unclear how the matrix M in (2.4) is to be selected. The principal difficulty when working with resticted or extended eigenvalue problems is the different orders of the involved matrices A and A_0. While in the modified eigenvalue problem the eigenvector matrices of A and $A_0 = A_1 \oplus A_2$ can be stored in the same memory locations, in the restricted eigenvalue problem case either extra storage has to be allocated to handle the increased size of A_0 or programming is becoming very cumbersome [1]. In the extended eigenvalue problem the order of A_0 is smaller than that of A, so there is no extra memory space needed. We can even hope for a speedup due to smaller ordered subproblems.

We chose to work in the setting of extended eigenvalue problems. As the theories for modified, restricted, and extended eigenvalue problems are essentially the same (cf. [3],[7]) we confine ourselves to review the latter in section 4.

3. The conquer part

The conquer part of the divide and conquer algorithm consists in the solution of the 'simple' eigenvalue problems $A_i\mathbf{x} = \lambda\mathbf{x}, i = 1,2$. These problems are simpler than (1.1) in the sense that the matrices A_i have lower order than A. They can be solved in the classical EISPACK way or by applying the divide and conquer algorithm again. The second approach is certainly favorable if a computer with more than two processors is available.

In the sequel we assume that we know the spectral decompositions

$$A_i = Q_i\Lambda_i Q_i^T, \qquad i = 1,2, \tag{3.1}$$

of the 'small' matrices A_1 and A_2 which make up A_0 in (2.14). The matrices $\Lambda_i = \mathrm{diag}(\lambda_0^{(i)}, \ldots, \lambda_{k_i-1}^{(i)})$ contain the eigenvalues of A_i in their diagonal. The columns of the orthogonal matrices $Q_i \in \mathbf{R}^{k_i \times k_i}$ are the corresponding eigenvectors.

4. The reassembly part

To obtain the spectral decomposition of $\bar{A}$ in (2.14) we make use of the techniques reviewed in [4] for low rank modified eigenvalue problems (see also [3],[7]). In this section we summarize the theory for extended eigenvalue problems. We start with

Lemma 4.1 (1) *Let $\lambda \notin \sigma(A_0)$, the spectrum of A_0. Then the matrices*

$$\bar{A} - \lambda = \begin{pmatrix} A_0 - \lambda & V \\ V^T & \Delta - \lambda \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} A_0 - \lambda & O \\ O^T & W(\lambda) \end{pmatrix}, \tag{4.1}$$

with

$$W(\lambda) = \Delta - \lambda - V^T(A_0 - \lambda)^{-1}V, \tag{4.2}$$

are congruent.

(2) *Let λ be an eigenvalue of A_0 of multiplicity μ and let $Q \in \mathbf{R}^{n\times\mu}$ be a matrix with orthonormal columns spanning the eigenspace $\mathcal{N}(A_0 - \lambda)$ corresponding to λ. Then the $(m+\mu)\times(m+\mu)$-matrices*

$$\begin{pmatrix} A_0 - \lambda & V & O \\ V^T & \Delta - \lambda & O \\ O & O & O_\mu \end{pmatrix} \tag{4.3a}$$

and

$$\begin{pmatrix} A_0 - \lambda & O & O \\ O & \Delta - \lambda - V^T(A_0 - \lambda)^+V & V^TQ \\ O & Q^TV & O_\mu \end{pmatrix} \tag{4.3b}$$

are congruent. In (4.3b), $(A_0 - \lambda)^+$ is the Moore-Penrose generalized inverse of $A_0 - \lambda$ [11],[20].

Proof. (1) First we see that

$$G^T \begin{pmatrix} A_0 - \lambda & V \\ V^T & \Delta - \lambda \end{pmatrix} G = \begin{pmatrix} A_0 - \lambda & O \\ O^T & W(\lambda) \end{pmatrix}$$

holds with

$$G = \begin{pmatrix} I_n & -(A_0-\lambda)^{-1}V \\ O^T & I_r \end{pmatrix}.$$

(2) We prove a slightly more general result than the one claimed in the second part of the Lemma. To that end let $\mathcal{U}(\lambda)$ be a neighborhood of the eigenvalue λ of $A_0\mathbf{x} = \lambda\mathbf{x}$ such that $\mathcal{U}(\lambda) \cap \sigma(A_0) = \{\lambda\}$ and define

$$(A_0-\xi)^\dagger := \begin{cases} (I-QQ^T)(A_0-\xi)^{-1}(I-QQ^T), & \xi \in \mathcal{U}(\lambda)\backslash\{\lambda\}, \\ (A_0-\lambda)^+, & \xi = \lambda. \end{cases}$$

Note that $I-QQ^T$ is the orthogonal projector on the orthogonal complement $\mathcal{R}(Q)^\perp$ of the subspace spanned by the columns of Q. Note furthermore that $\xi \mapsto (A_0-\xi)^\dagger$ is a *continuous* matrix function on $\mathcal{U}(\lambda)$ [11, p.218].

One verifies that for all $\xi \in \mathcal{U}(\lambda)$

$$(A_0-\xi)^\dagger(A_0-\xi)\mathbf{x}_j = (A_0-\xi)(A_0-\xi)^\dagger\mathbf{x}_j = \begin{cases} \mathbf{x}_j, & A_0\mathbf{x}_j = \lambda_j\mathbf{x}_j, \mathbf{x}_j \in \mathcal{R}(Q)^\perp, \\ \mathbf{0}, & \mathbf{x}_j \in \mathcal{R}(Q). \end{cases}$$

So we have

$$\begin{aligned} &G_e^T \begin{pmatrix} A_0-\xi & V & O \\ V^T & \Delta-\xi T & O \\ O & O & (\lambda-\xi)I_\mu \end{pmatrix} G_e \\ &= \begin{pmatrix} A_0-\xi & O & O \\ O & \Delta-\xi T - V^T(A_0-\xi)^\dagger V & V^TQ \\ O & Q^TV & (\lambda-\xi)I_\mu \end{pmatrix} \end{aligned} \tag{4.4}$$

for all $\xi \in \mathcal{U}(\lambda)$ where

$$\begin{aligned} G_e &= \begin{pmatrix} I_n-QQ^T & O & Q \\ O & I_r & O \\ Q^T & O & O_\mu \end{pmatrix} \begin{pmatrix} I_n & -(A_0-\xi)^\dagger V & O \\ O & I_r & O \\ O & O & I_\mu \end{pmatrix} \\ &= \begin{pmatrix} I_n-QQ^T & -(A_0-\xi)^\dagger V & Q \\ O & I_r & O \\ Q^T & O & O_\mu \end{pmatrix}. \end{aligned}$$

Setting $\xi := \lambda$ yields the claimed result. □

As in [7] we call the matrix $W(\lambda)$ in (4.2) *Weinstein matrix,* a name in common use in the theory of the methods of intermediate problems [33]. In structural engineering $W(\lambda)$ is known as *Kron's matrix* [30]. $W(\lambda)$ is the Schur complement of $A_0-\lambda$ in $\bar{A}-\lambda$. Its determinant $\det W(\lambda)$ is called Weinstein determinant or Kron's determinant. Since $\det G = 1$ for any $\lambda \notin \sigma(A_0)$ we have

$$\det W(\lambda) = \frac{\det(A-\lambda)}{\det(A_0-\lambda)} = \frac{\det(\bar{A}-\lambda)}{\det(A_0-\lambda)}, \quad \lambda \notin \sigma(A_0). \tag{4.5}$$

So, $\det W(\lambda)$ has singularities at the eigenvalues of A_0 and vanishes at the eigenvalues of A which are not at the same time eigenvalues of A_0. From (4.4) we see that

$$\det W_e(\lambda) = \lim_{\xi\to\lambda} \det W(\lambda)(\lambda-\xi)^\mu. \tag{4.6}$$

If we make use of the spectral decompositions (3.1) of A_1 and A_2, the Weinstein matrix in (4.2) becomes

$$W(\lambda) = \Delta - \lambda + U^T \left((\Lambda_1 - \lambda I_{k_1}) \oplus (\Lambda_2 - \lambda I_{k_2})\right)^{-1} U, \quad U = \left(Q_1^T \oplus Q_2^T\right) V. \quad (4.7)$$

Note that for the calculation of U only the last r rows of Q_1 and the first r rows of Q_2 need to be known. As $W(\lambda)$ is symmetric, its computation costs $nr(r+3)+O(r^3)$ flops provided that U is known. [1] When forming $W(\lambda)$, $2r(n-r)$ flops are needed to compute the auxiliary vectors $\mathbf{h}_i := ((\Lambda_1 - \lambda) \oplus (\Lambda_2 - \lambda))^{-1} \mathbf{u}_i$. The rest of the work is essentially spent in forming the scalar products $\mathbf{h}_i^T \mathbf{u}_j$ of the auxiliary vectors with the columns $\mathbf{u}_j$ of U.

We denote the submatrix

$$W_e(\lambda) = \begin{pmatrix} \Delta - \lambda - V^T (A_0 - \lambda)^+ V & V^T Q \\ Q^T V & O_\mu \end{pmatrix} \quad (4.8)$$

of the matrix in (4.3b) as the *extended Weinstein matrix*. As we have seen in the proof of Lemma 4.1 we can define the extended Weinstein matrix in a neighborhood $\mathcal{U}(\lambda)$ of the eigenvalue λ if we replace $(A_0 - \lambda)^+$ by $(A_0 - \lambda)^\dagger$. This is useful if the Weinstein matrix has to be evaluated in the neighborhood of an eigenvalue of A because its elements have poles in λ. In a similar way several poles of the determinant of the Weinstein matrix could be removed [3].

A basis of $\mathcal{N}(A_0 - \lambda)$ is obtained by selecting those columns of $Q_1 \oplus Q_2$ which are eigenvectors of A_0 corresponding to λ. Making use of the spectral decompositions (3.1), equation (4.8) can be transformed into

$$W_e(\lambda) := \begin{pmatrix} \Delta - \lambda + U^T (\Lambda_1 \oplus \Lambda_2 - \lambda)^+ U & U^T E \\ E^T U & O \end{pmatrix}, \quad (4.9)$$

where $E = (Q_1^T \oplus Q_2^T)Q$. The columns of E are axis vectors if the basisvectors $\mathbf{q}_i$ of $\mathcal{N}(A_0 - \lambda I)$ are selected among the columns of $(Q_1^T \oplus Q_2^T)$. In this case $E^T U$ simply consists of some of U's rows. Note that the computation of the extended Weinstein matrix is no more expensive than its ordinary counterpart. However the computation of its inertia is.

The following Theorem is obtained if Sylvester's law of inertia [20, p.274] is applied on the congruent matrices in Lemma 4.1.

Theorem 4.2 *$N(\lambda)$, the number of eigenvalues of A which are $< \lambda$, is given by*

$$N(\lambda) = \begin{cases} N_0(\lambda) + \nu(W(\lambda)), & \lambda \notin \sigma(A_0), \\ N_0(\lambda) + \nu(W_e(\lambda)), & \lambda \in \sigma(A_0), \end{cases} \quad (4.10)$$

where $N_0(\lambda)$ denotes the number of eigenvalues of A_0 less than λ and $\nu(.)$ means the negative inertia of the involved matrix. □

[1] A *flop* is a floating point operation, i.e. a floating point add, subtract, multiply or divide [20, p.19].

The interlacing properties (2.15) are a simple consequence of this Theorem.

Theorem 4.2 permits to find the eigenvalues of $Ax = \lambda x$ by spectrum slicing methods as e.g. the bisection algorithm [6].

We turn now to the eigenvectors of A. In view of (2.14), an eigenvector of A is obtained from an eigenvector of $\bar{A}$ by a permutation of components. We describe how to obtain the latter. We assume first that $\lambda \notin \sigma(A_0)$. We make use of the fact that G maps the nullspace of $(A_0 - \lambda) \oplus W(\lambda)$ onto the nullspace of $\bar{A} - \lambda$. Now,

$$\mathbf{y} := \begin{pmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{pmatrix} := G \begin{pmatrix} \mathbf{0} \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} -(A_0 - \lambda)^{-1} V \mathbf{x}_2 \\ \mathbf{x}_2 \end{pmatrix}, \quad \mathbf{x}_2 \in \mathcal{N}(W(\lambda)). \tag{4.11}$$

Therefore, (4.11) yields a basis of $\mathcal{N}(\bar{A} - \lambda)$ provided that we know a basis of $\mathcal{N}(W(\lambda))$. With the spectral decompositions (3.1) we get

$$\mathbf{y}_1 = -(Q_1 \oplus Q_2)(\Lambda_1 \oplus \Lambda_2 - \lambda I)^{-1} U \mathbf{x}_2, \qquad \mathbf{y}_2 = \mathbf{x}_2. \tag{4.12}$$

The computation of $\mathbf{y}$ according to (4.11)–(4.12) costs $n^2 + n + O(r^2)$ flops. The matrix $\times$ vector product of $Q_1 \oplus Q_2$ with $(\Lambda_1 \oplus \Lambda_2 - \lambda I)^{-1} U \mathbf{y}$, which has to be formed n times, once for every eigenvector of $\bar{A}$, is the most expensive part of the divide and conquer algorithm. It gives rise to the serial $O(n^3)$ complexity of the algorithm.

Let now $\lambda \in \sigma(A_0)$. G_e maps the nullspace of the matrix in (4.3b) one-to-one onto the nullspace of the matrix in (4.3a). But the former consists of two mutually orthogonal components: one consisting of vectors of the form $\begin{pmatrix} \mathbf{x}_1 \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}$, $\mathbf{x}_1 \in \mathcal{R}(Q) = \mathcal{N}(A_0 - \lambda)$, and one consisting of vectors of the form $\begin{pmatrix} \mathbf{0} \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix}, \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} \in \mathcal{N}(W_e(\lambda)) \subset \mathbf{R}^{r+\mu}$. The image of the first component under G_e is $\begin{pmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{y}_3 \end{pmatrix}, \mathbf{x}_1 = Q\mathbf{y}_3$, $\mathbf{y}_3 \in \mathbf{R}^{\mu}$. G_e is restricted to $\{\mathbf{0}\} \times \mathcal{N}(W_e(\lambda))$ yields a bijective map from $\mathcal{N}(W_e(\lambda))$ onto $\mathcal{N}(\bar{A} - \lambda)$,

$$\begin{aligned} \begin{pmatrix} \mathbf{x}_2 \\ \mathbf{x}_3 \end{pmatrix} \longmapsto & \begin{pmatrix} (\lambda - A_0)^{+} V \mathbf{x}_2 + Q \mathbf{x}_3 \\ \mathbf{x}_3 \end{pmatrix} \\ & = \begin{pmatrix} (Q_1^T \oplus Q_2^T)\left[(\lambda - \Lambda_1 \oplus \Lambda_2)^{+} U \mathbf{x}_2 + E \mathbf{x}_3\right] \\ \mathbf{x}_3 \end{pmatrix}. \end{aligned} \tag{4.13}$$

The cost of the computation of $(\mathbf{x}_2^T, \mathbf{x}_3^T)^T$ in (4.13) is about as high as that of the computation of $\mathbf{y}$ in (4.11) as, due its special structure, forming $E\mathbf{x}_3$ is practically for free.

5. Insertion: The generalized eigenvalue problem

Instead of (1.1) we may consider the *generalized* eigenvalue problem

$$A\mathbf{x} = \lambda M \mathbf{x}, \tag{5.1}$$

where M is a banded, positive definite matrix. Without loss of generality, we can assume the bandwidth of both A and M to be equal. (Otherwise, we 'increase' the bandwidth of the matrix with the narrower band up to the bandwidth of the matrix with the wider band by inserting zero off-diagonals.) The generalized *tridiagonal* eigenvalue problem has been investigated by Beattie and Ribbens [8].

As in subsection 2.3 we can define (cf. (2.14))

$$\bar{A} = P^T A P = \begin{pmatrix} A_0 & V \\ V^T & \Delta \end{pmatrix}, \qquad \bar{M} = P^T M P = \begin{pmatrix} M_0 & F \\ F^T & \Phi \end{pmatrix}. \tag{5.2}$$

The first part of Lemma 4.1 can be transcribed into the following form.

Lemma 5.1 *Let $\lambda \notin \sigma(A_0; M_0)$, the spectrum of A_0 relative to M_0. Then the matrices*

$$\bar{A} - \lambda\bar{M} = \begin{pmatrix} A_0 - \lambda M_0 & V - \lambda F \\ V^T - \lambda F^T & \Delta - \lambda\Phi \end{pmatrix} \quad \textit{and} \quad \begin{pmatrix} A_0 - \lambda M_0 & O \\ O^T & W(\lambda) \end{pmatrix}, \tag{5.3}$$

with

$$W(\lambda) = \Delta - \lambda\Phi - (V - \lambda F)^T (A_0 - \lambda)^{-1}(V - \lambda F), \tag{5.4}$$

are congruent. □

The second part of the Lemma can be transcribed in a similar way. In the proof we define the matrix $(A_0 - \xi M_0)^\dagger$ by

$$(A_0 - \xi M_0)^\dagger := \begin{cases} (I - QQ^T M_0)(A_0 - \xi M_0)^{-1}(I - M_0 QQ^T), & \xi \in \mathcal{U}(\lambda)\backslash\{\lambda\}, \\ (A_0 - \lambda M_0)^+, & \xi = \lambda. \end{cases}$$

Note that $I - QQ^T M_0$ is the M_0-orthogonal projector on the orthogonal complement $\mathcal{R}(Q)^\perp$ of the subspace spanned by the columns of Q. $I - M_0 QQ^T$ is the M_0^{-1}-orthogonal projector on $(A - \xi M_0)\mathcal{R}(Q)^\perp$ (which is invariant under changes of $\xi \in \mathcal{U}(\lambda)$.

So, with the modifications described in this section, our algorithm can be used for the generalized eigenvalue problem (5.1) in just the way it is used for solving the special eigenvalue problem (1.1).

6. Two algorithms

A divide and conquer algorithm has the structure depicted in Fig. 6.1

If we assume that the matrix splitting has negligible cost, the algorithm consists of three processes. Process 0 performs the reassembly part. It *depends* on the output of processes 1 and 2 which compute the spectral decomposition of A_1 and A_2. That means that process 0 can start to work only if processes 1 and 2 both have completed. The arrows in Fig. 6.1 indicate the data dependency among the three processes involved.

If the split eigenvalue problems are treated by the same divide and conquer algorithm, we end up with a dependency graph which is a balanced binary tree.

Fig. 6.2 shows an example of a binary tree of height $q = 2$. We denote a node in the tree by a pair of nonnegative integers (l, m). The first integer l indicates

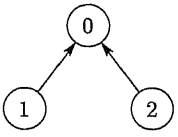

Figure 6.1: Dependency graph for divide and conquer primitive.

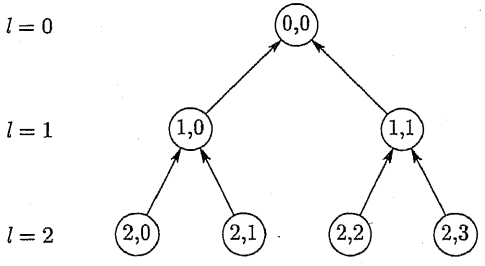

Figure 6.2: Binary tree of height $q = 2$.

the level of the node; the root has level 0 while the leaves have level q. Nodes of equal level are numbered from 0 to $2^l - 1$ such that (l, m) has sons $(l + 1, 2m)$ and $(l + 1, 2m + 1)$. If the order of the overall problem (1.1) is $n = 2^q(k + r) - r$ and splittings are performed so that the split eigenvalue problems have equal order then the order of the eigenvalue problem $A_{l,m}\mathbf{x} = \lambda\mathbf{x}$ that is to be solved in a node of level l is $n_l := 2^{q-l}(k + r) - r = (n - (2^l - 1)r)/2^l$.

We formulate the divide-and-conquer algorithm in the following way:

ALGORITHM D_and_C (A, n, r, q).
{ This algorithm computes the spectral decomposition of the band matrix A of order $n = 1$ and band width r by a divide-and-conquer algorithm. The matrix A is thereby split q times recursively. }

Define $S := \left\{(l, m)\middle| 0 \le l \le q, 0 \le m < 2^l\right\} \subset \mathbb{N}_0^2$, the set of admissible index pairs.
while $|S| > 0$,
  Pick $(l, m) \in S$ at random.
  if $l = q$,
    Compute the spectral decomposition $A_{q,m} = Q_{q,m}\Lambda_{q,m}Q_{q,m}^T$ of $A_{q,m} \in \mathbb{R}^{k\times k}$ by the combined the EISPACK subroutines `bandr` and `tql2`.

Remove (l,m) from S.
else $\{l < q\}$
if $(l+1,2m) \notin S \wedge (l+1,2m+1) \notin S$,
Compute the spectral decomposition

$$\bar{A}_{l,m} = \bar{Q}_{l,m}\Lambda_{l,m}\bar{Q}_{l,m}^T$$

of the matrix

$$\bar{A}_{l,m} = \begin{pmatrix} A_{l+1,2m} \oplus A_{l+1,2m+1} & V_{l,m} \\ V_{l,m}^T & \Delta_{l,m} \end{pmatrix} \in \mathbf{R}^{n_l \times n_l}$$

by Algorithm Assemble($A_{l,m}$, $A_{l+1,2m}$, $A_{l+1,2m+1}$, $\Delta_{l,m}$, $V_{l,m}$) below.
$Q_{l,m} := P_{l,m}\bar{Q}_{l,m}$. { Whence, $A_{l,m} = Q_{l,m}\Lambda_{l,m}Q_{l,m}^T$ }
$S := S \setminus \{(l,m)\}$.
end
end
end

Algorithm D_and_C was formulated in view of of SCHEDULE, a tool to facilitate writing parallel Fortran programs written by Dongarra and Sorensen [15]. Notice that any two tasks (l,m) which satisfy $(l+1,2m) \notin S$ and $(l+1,2m+1) \notin S$ are independent of each other and can thus be executed in parallel.

Remarks. (1) Maps from S onto $\mathbf{N}_0$ and back are given by

$$\begin{array}{rcl} S & \longrightarrow & \mathbf{N}_0 \\ (l,m) & \longmapsto & 2^l + (m-1) \end{array}$$

and

$$\begin{array}{rcl} \mathbf{N}_0 & \longrightarrow & S \\ n & \longmapsto & (\lfloor \log_2(n+1) \rfloor, (n+1) \bmod \lfloor \log_2(n+1) \rfloor) \end{array}$$

(2) Ipsen and Jessup [21] numbered the tasks or nodes in the binary tree in a similar way as we did. In their notation however the value of l *decreases* as the level of the node increases. □

The difficult part in a divide and conquer algorithm – at least in linear algebra problems – is the assembly of the small problems to solve the next larger problems. We describe in the sequel two possible algorithms. In the first one we update the spectral decomposition of A in one step. In the second algorithm we get from A_0 to A in r steps. In each step the matrix order is increased by only one. This corresponds to the modification (2.2) of Dongarra and Sorensen. It turns out that the complexities of the two algorithms are completely different.

6.1. The rank-r extension algorithm

An algorithm for this part may in our case have the following form:

ALGORITHM Assemble (A, A_1, A_2, Δ, V).
{ This algorithm computes the spectral decomposition of

$$A = \begin{pmatrix} A_0 & V \\ V^T & \Delta \end{pmatrix}, \qquad A_0 = A_1 \oplus A_2,$$

making use of the known spectral decompositions $A_i = Q_i \Lambda_i Q_i^T$, $i = 1, 2$. }

Merge and sort the eigenvalues $\lambda_j^{(1)}$ and $\lambda_j^{(2)}$ of A_1 and A_2 into

$$\lambda_0^{(0)} \le \lambda_1^{(0)} \le \cdots \le \lambda_{n-1}^{(0)}.$$

(These are the increasingly ordered eigenvalues of $A_0 = A_1 \oplus A_2$.)
Set $m := |\sigma(A_0)|$. Let

$$\hat{\lambda}_0^{(0)} < \hat{\lambda}_1^{(0)} < \cdots < \hat{\lambda}_{m-1}^{(0)}$$

denote the elements of $\sigma(A_0)$.
Determine $\hat{\lambda}_{-1}^{(0)}$ and $\hat{\lambda}_m^{(0)}$ such that $\hat{\lambda}_{-1}^{(0)} \le \lambda_j \le \hat{\lambda}_m^{(0)}$ for all $j = 0, \ldots, n-1$.
$U := (Q_1 \oplus Q_2)^T V$.
for $j = 0 : m-1$,
  Compute the extended Weinstein matrix $W_e(\hat{\lambda}_j^{(0)})$ and its inertia (π_j, ν_j, ζ_j).
  if $W_e(\hat{\lambda}_j^{(0)})$ is singular,
    Determine $\mathcal{N}(W_e(\hat{\lambda}_j^{(0)}))$.
    Determine an orthonormal basis of $\mathcal{N}(A - \hat{\lambda}_j^{(0)} I)$, the eigenspace of A corresponding to $\hat{\lambda}_j^{(0)}$ according to (4.13).
  end
end
for $j = 0 : m$, { Compute the eigenvalues in the open interval $(\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)})$ }
  $M := N_0(\hat{\lambda}_j^{(0)}) + \nu_j - N_0(\hat{\lambda}_{j-1}^{(0)}) - \nu_{j-1} - \zeta_{j-1}$, { $M = |(\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)}) \cap \sigma(A)|$ }
  if $M > 0$,
    (i) Isolate the eigenvalues in the interval by bisection using the inertia of the Weinstein matrix $W(\lambda) = \Delta - \lambda - V^T(A_0 - \lambda I)^{-1} V$.

    (ii) Determine the isolated eigenvalues by an appropriate (superlinearly convergent) zerofinder applied to $\det W(\lambda) = 0$.

    (iii) Compute the corresponding eigenvectors according to (4.12) and (2.14).
  end
end

Note that the tasks in both for-loops are independent of each other and can be computed in parallel. That task computing the eigenvalues in interval $(\hat{\lambda}_{j-1}^{(0)}, \hat{\lambda}_j^{(0)})$ is data dependent exactly on the two tasks which deal with the interval endpoints $\hat{\lambda}_{j-1}^{(0)}$ and $\hat{\lambda}_j^{(0)}$, respectively. So, with the notation in Fig. 6.1, the process which executes procedure Assemble can spawn several subprocesses (tasks) which are data dependent among each other in the way depicted in Fig. 6.3.

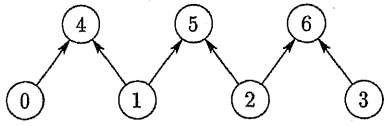

Figure 6.3: Dependency graph for subprocesses.

In EISPACK, the banded matrix is first transformed into a similar tridiagonal matrix. The resulting eigenvalue problem is then solved by the cubically convergent tridiagonal QR algorithm [20, §8.2]. The transformation into tridiagonal form can be performed in two different ways with EISPACK. Either the subroutine `bandr` is used which exploits the banded structure of the matrix by using Givens rotations [29] or subroutine `tred2` is applied which transforms any symmetric into a similar tridiagonal matrix using Householder reflectors [20, p.419]. The *serial complexities* of these two EISPACK approaches are

$$\begin{aligned} Z_{\texttt{bandr/tql2}}(n,r) &\cong n^2\tfrac{r-1}{r}(6n+12r) && \text{(tridiagonalization by bandr)} \\ &\quad +9n^2(n-1) && \text{(tridiagonal QR algorithm)} \\ &\cong (6\tfrac{r-1}{r}+9)n^3+(12(r-1)-9)n^2 \text{ flops,} \end{aligned}$$

and

$$\begin{aligned} Z_{\texttt{tred2/tql2}}(n,r) &\cong \tfrac{8}{3}n^3-4n^2 && \text{(tridiagonalization by tred2)} \\ &\quad +9n^2(n-1) && \text{(tridiagonal QR algorithm)} \\ &\cong \tfrac{35}{3}n^3-13n^2 \text{ flops,} \end{aligned}$$

respectively. The break even point of these algorithms is very small! Numerical experiments indicate that `bandr` is faster than `tred2` for at most $r = 2$. `bandr` as well as `tred2` can be vectorized and/or parallelized [23],[5],[17]. Although the complexity analysis is unfavorable for `bandr`, we did not observe big differences in speed and accuracy for small r. Recall that `tred2` does not take advantage of the band structure. Therefore, A has to be stored as a full matrix if `tred2` is to be applied.

Remark. The situation is more favorable for `bandr` if only the eigenvalues are to be computed. In this case `tred1` (= `tred2` without accumulation of the Householder reflectors) costs $\sim\frac{4}{3}n^3$ vs. $\sim 12rn^2$ flops for `bandr`. So the break even point is around $r = n/9$. □

For the complexity analysis of the divide and conquer algorithm we set $p = 2^q$, where q is again the height of the dependency graph (cf. Fig. 6.2). To facilitate the analysis we do not take into account that the order of the extended Weinstein matrices are $> r$. Furthermore, we assume that in the average s evaluations of a Weinstein matrix are necessary to locate an eigenvalue to the desired accuracy. Thus for the divide and conquer algorithm with highest level q we obtain the following

serial complexity, [2]

$$\begin{aligned} Z_{\mathrm{d\&c}}^{(1\times r)}(n,r,p,s) &\cong pn_q^2(\tfrac{35}{3}n_q - 13) && (\texttt{tred2/tql2 on level } q \text{ nodes}) \\ &\quad + \sum_{l=0}^{q-1} 2^l 2n_l r^2 && (U = (Q_1 \oplus Q_2)^T V) \\ &\quad + s\sum_{l=0}^{q-1} 2^l n_l[n_l r(r+3)] && (\text{Weinstein matrices}) \\ &\quad + \sum_{l=0}^{q-1} 2^l n_l[n_l^2 + n_l] && (\text{eigenvectors}) \\ &\cong \tfrac{n^3}{3p^2}(4(p^2-1)+35) + \tfrac{n^2}{p}(2(p-1)(sr^2+3r+1)-13) \text{ flops} \end{aligned}$$

The coefficient of the n^3 term quickly tends to $\frac{4}{3}$ as p goes to ∞. For $p = 8$ the coefficient is $\approx \frac{4.5}{3}$. For $p > 1$ we have

$$Z_{\mathrm{d\&c}}^{(1\times r)}(n,r,p,s)/Z_{\texttt{tred2/tql2}}(n,r) \quad \longrightarrow \quad \frac{4}{35}\frac{p^2-1}{p^2} + \frac{1}{p^2} < 1 \qquad (n \to \infty)$$

provided that s/n is going to zero with growing n. This means that the divide and conquer algorithm is faster than the classical algorithm for *any* r if n is sufficiently large. For small n however, the n^2 and n^3 terms may well be of the same order of magnitude. We observed in our computational experiments values of $s \cong 10$. So, if $r = 3$, $p = 8$, and $s = 10$ the n^2 term of $Z_{\mathrm{d\&c}}^{(1\times r)}$ outweighs the n^3 term for $n < 28$. In Tab. 6.1 the values n are listed for which $Z_{\mathrm{d\&c}}^{(1\times r)} = Z_{\texttt{tred2/tql2}}$ for several r assuming $p = 8$ and $s = 10$. n is practically independent of p. A least squares fit showed that $n(r) \approx 1.2 + 1.7r^2$.

Table 6.1: n such that $Z_{\mathrm{d\&c}}^{(1\times r)}(n,r,8,10) = Z_{\texttt{tred2/tql2}}(n,r)$

$r =$	3	4	6	8	10	12	16	20	24
$n(r) =$	17	29	63	111	173	249	441	689	993

If we compare the band divide and conquer algorithm with `tql2` combined with Cuppen's divide and conquer algorithm for tridiagonal matrices, we get the numbers depicted in Table 6.1. A least squares fit yields $n(r) \approx -5.5 + 6.7r^2$.

Comparing the two algorithms for increasing orders n we get

$$Z_{\mathrm{d\&c}}^{(1\times r)}(n,r,p,s)/Z_{\texttt{tred2/Cuppen}}(n,r) \quad \longrightarrow \quad \frac{4(p^2-1)+35}{12(p^2-1)+35} \qquad (n \to \infty)$$

which is about $\frac{1}{3}$ already for small values of p. The cross points of the complexity curves are here much higher than before. The numbers indicate that it is only

[2] Assuming $n_l = n/2^l$, we have

$$\sum_{l=0}^{q-1} 2^l n_l = \sum_{l=0}^{q-1} n = qn, \qquad \sum_{l=0}^{q-1} 2^l n_l^{k+1} = n^{k+1}\sum_{l=0}^{q-1}\frac{1}{2^{lk}} = \frac{2^k}{2^k-1}\frac{p^k-1}{p^k}n^{k+1}, \quad 2^q = p, \quad k > 0$$

Table 6.2: n such that $Z_{\text{d\&c}}^{(1\times r)}(n,r,8,10) = Z_{\texttt{tred2/Cuppen}}(n,r)$

$r=$	3	4	6	8	10	12	16	20	24
$n(r)=$	55	101	235	421	661	955	1701	2661	3835

reasonable to use the band divide and conquer algorithm if the semi-bandwidth r is small, a fact which isn't astonishing at all.

If we assume that we have n_{proc} processors to our disposal and n is reasonably large, we can obtain further speedups of about n_{proc} due to the inherent parallelism of the algorithm. This means a further gains of the divide and conquer algorithms compared to the EISPACK routines. The relation between the divide and conquer alternatives should not change very much as the tridiagonalization is parallelizable too.

Remark. If only the eigenvalues of $A\mathbf{x} = \lambda\mathbf{x}$ are desired we need to compute only r components of the vector $\mathbf{x}$ in (4.8). This reduces the complexity of the divide and conquer algorithm to about $n^2(\frac{p-1}{p}(2sr^2+8r)+18r/p)$ flops. The combined `bandr` and `tql1` need $\sim 12rn^2$ flops for this task. □

6.2. The rank-1 update algorithm

Now we describe an alternative to the above Algorithm Assemble, which computes the spectral decomposition $\bar{A} = Q\Lambda Q^T$ of $\bar{A} = \begin{pmatrix} A_0 & V \\ V^T & \Delta \end{pmatrix}$. But instead of directly working with the entire 'rank-r extension' we proceed here in steps of rank one. This corresponds to writing a rank-r modification as the sum of r rank-1 matrices (cf. (2.3)). Working with rank-1 extensions would facilitate the zerofinding process and would permit the application of the simple deflation technique of Cuppen's algorithm [10],[12],[14].

Without loss of generality we assume that Δ is diagonal, $\Delta = \text{diag}(\delta_1,\ldots,\delta_r)$. Let $V = [\mathbf{v}_1,\ldots,\mathbf{v}_r]$. Then we define the auxiliary matrices

$$\bar{A}_k := \begin{pmatrix} \bar{A}_{k-1} & \mathbf{v}_k \\ \mathbf{v}_k^T & \delta_k \end{pmatrix}, \qquad k = 0,\ldots,r.$$

Assume that we have already computed the spectral decomposition

$$\bar{A}_{k-1} = Q_{k-1}\Lambda_{k-1}Q_{k-1}^T$$

of $\bar{A}_{k-1}$. Then

$$\bar{A}_k = \begin{pmatrix} Q_{k-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} \begin{pmatrix} \Lambda_{k-1} & \mathbf{u}_k \\ \mathbf{u}_k^T & \delta_k \end{pmatrix} \begin{pmatrix} Q_{k-1}^T & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}, \qquad \mathbf{u}_k = Q_{k-1}^T\mathbf{v}_k.$$

The matrix in the middle is a so-called arrow matrix. Let

$$\begin{pmatrix} \Lambda_{k-1} & \mathbf{u}_k \\ \mathbf{u}_k^T & \delta_k \end{pmatrix} = S_k\Lambda_k S_k{}^T \tag{6.1}$$

be its spectral decomposition. Then

$$\bar{A}_k = \begin{pmatrix} Q_{k-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} S_k \Lambda_k {S_k}^T \begin{pmatrix} Q_{k-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}^T = Q_k \Lambda_k {Q_k}^T, \quad Q_k := \begin{pmatrix} Q_{k-1} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix} S_k,$$

is the spectral decomposition of $\bar{A}_k$. In each of the r extension steps, the solution of the eigenvalue problem (6.1) is the principal task. This problem is well-known and can be tackled as the divide and conquer algorithm for the symmetric tridiagonal eigenvalue problem [27]. As the Weinstein matrix is a scalar the inequalities (2.15) become very simple. By Theorem 4.2 we see that the inequalities

$$\lambda_{j-1}^{(k-1)} \leq \lambda_j^{(k)} \leq \lambda_j^{(k-1)}, \qquad 0 < j < n-1,$$

hold.

The complexity analysis shows that an algorithm incorporating the just described extension strategy would need

$$Z_{\mathrm{d\&c}}^{(r\times 1)}(n,r,p,s) \cong \frac{n^3}{3p^2}(4(p^2-1)(2r-1)+35) + \frac{n^2}{p}(2r(p-1)(6s+1)-13) \text{ flops}$$

to compute the spectral decomposition of $\bar{A} = \bar{A}_k$. To compute only the eigenvalues, $2rn^2/p(6s+2r+9/p)+O(n)$ flops suffice. Here, again, s means the average number of iteration steps needed to find a zero of the secular equation. This s can be expected to be smaller than the s in the $1 \times r$ algorithm since we have here a quadratically convergent zero finder. The main difference between the two divide and conquer algorithms are the $\frac{4}{3}(2r-1)$ and $\frac{4}{3}$ terms in the $1 \times r$ and the $r \times 1$ algorithm, respectively. The difference comes from the updating of the eigenvectors. While in the algorithm of the previous section Q_0, which is the direct sum of two smaller matrices, is multiplied *once* with the n eigenvectors of the interior eigenproblem, in this algorithm a *sequence* of matrices Q_k is formed by $Q_k := (Q_{k-1} \oplus 1)S_k$ whereby all Q_k, $k > 0$, and S_k are full.

7. Computational results

We implemented the above described rank-r extension divide and conquer algorithm on an Alliant FX-80, a shared memory MIMD computer with 6 processors, each with vector facilities. The tasks of the algorithm have been distributed on the processors with the aid of SCHEDULE, a tool to ease writing parallel programs [15], [16]. The input to SCHEDULE is essentially the graph describing the data-dependencies among the tasks of the program. This is in our case the binary tree of Fig. 6.2. SCHEDULE is then capable of distributing the independent tasks on the processors available. SCHEDULE does this in quite the way algorithm D_and_C was formulated.

The tasks corresponding to the leaves of the tree invoke the EISPACK routines `bandr` and `tql2` to solve the smallest eigenvalue problems. The tasks associated with the remaining nodes execute algorithm Assemble.

The tasks corresponding to the tree's leaves give rise to some parallelism. As the number of nodes per level decreases rapidly when approaching the root, this

parallelism is not sufficient for a satisfactory implementation of the divide and conquer algorithm on a multiprocessor computer. However, as mentioned right after the formulation of algorithm Assemble, the determination of the zeros of the Weinstein determinant and the computation of the corresponding eigenvectors in different intervals are independent of each other. Therefore we form subtasks by grouping the intervals. As a particular interval belongs to only one set of intervals, the subtasks are mutually independent and can be executed in parallel by different processors. The generation of these subprocesses during the program execution is done by SCHEDULE by a process called 'dynamic spawning' [15].

In our numerical experiments we built up trees of height $q = 3$ (cf. Fig 6.2). So, our tree had $p = 2^q = 8$ leaves and as many associated parallel tasks. This number was chosen such that all six processors of the Alliant were running from the beginning of the calculation. The number of subtasks we formed in the lower level nodes was 12 *per level.* Of course, the estimated sizes of the subtasks of one level were about equal. By forming twice as many tasks per tree level as there were processors, we hoped to get a satisfactory load balance. The speedups listed in Table 7.1 indicate that we were quite successful in this respect.

To avoid huge function values when computing zeros of the Weinstein determinant close to one of its poles, we replaced $\det W(\lambda)$ by

$$f(\lambda) := (\lambda_{j-1}^{(0)} - \lambda)^{\mu_{j-1}} (\lambda_j^{(0)} - \lambda)^{\mu_j} \det W(\lambda), \qquad \lambda_{j-1}^{(0)} < \lambda < \lambda_j^{(0)}. \tag{7.1}$$

Here, μ_k denotes the multiplicity of $\lambda_j^{(0)}$ as eigenvalue of A_0. We sped up bisection with the zerofinder `zeroin` [9]. `zeroin` is a combination of bisection, linear interpolation and quadratic inverse interpolation. The superlinearly convergent zerofinder is invoked as soon as an interval containing precisely one zero of $f(\lambda)$ was encountered. In every iteration step, `zeroin` not only updates the approximation to the zero but also two further values which bracket the root. So, convergence is guaranteed to be towards the correct zero.

We tested the behavior of our algorithm by means of four test problems of orders n ranging from 100 to 300 with semi-bandwidth $r = 3$ or $r = 6$. The matrices were generated by a random number generator. The tests were run at times when no one else was active on the machine. The times we measured for these test problems are listed in Table 7.1. The times are the best of several (usually four) runs.

We compared our algorithm with the two EISPACK solution paths `bandr/tql2` and `tred2/tql2` [31], [18]. Notice that the three algorithms have different storage requirements. `bandr/tql2` needs (essentially) space for a full matrix and a $n \times (r+1)$ block matrix to store the eigenvector matrix Q and the band matrix A, respectively. `tred2/tql2` needs space for two full matrices. Our algorithm utilizes only a $n \times (r+1)$ block for A, but needs two full matrices for the eigenvector matrices corresponding to different levels in the binary tree.

Table 7.1 illustrates the predicted behavior of the algorithm. `tred2/tql2` is faster than `bandr/tql2` for $r = 6$. The differences are minor, however.

The divide and conquer algorithm is slower than both EISPACK combinations for the smaller problems with $n \leq 200$, the differences becoming bigger with increasing bandwidth. When – for fixed r – the matrix order n is chosen sufficiently large, the new algorithm becomes faster than the EISPACK competitors.

Problem size	Number of processors	Elapsed times [seconds]			Speedups	
(n,r)	p	`bandr/ tql2`	`tred2/ tql2`	new algorithm	new vs. new($p=1$)	new vs. EISPACK
(100,3)	1	2.53	2.59	3.65	1.00	0.69
(100,3)	2			1.91	1.91	1.32
(100,3)	3			1.34	2.72	1.88
(100,3)	4			1.09	3.35	2.32
(100,3)	5			0.92	4.01	2.75
(100,3)	6			0.79	4.65	3.20
(100,6)	1	2.95	2.60	8.28	1.00	0.31
(100,6)	2			4.35	1.90	0.60
(100,6)	3			3.04	2.73	0.85
(100,6)	4			2.42	3.42	1.07
(100,6)	5			2.10	3.95	1.24
(100,6)	6			1.73	4.78	1.50
(200,6)	1	19.54	18.21	24.36	1.00	0.74
(200,6)	2			12.94	1.88	1.41
(200,6)	3			8.88	2.74	2.05
(200,6)	4			6.97	3.50	2.61
(200,6)	5			6.61	3.69	2.76
(200,6)	6			5.12	4.76	3.55
(300,6)	1	63.17	62.04	52.07	1.00	1.19
(300,6)	2			27.34	1.91	2.26
(300,6)	3			18.75	2.77	3.31
(300,6)	4			14.44	3.60	4.29
(300,6)	5			13.54	3.84	4.58
(300,6)	6			10.53	4.95	5.89

Table 7.1: Timings and speedups

The parallelization behavior is satisfactory. We obtain speedups up to almost 5 with 6 processors. This means an efficiency of above 80%. We note here, that the EISPACK routines are parallelizable as well. The largest test problem indicates however that our algorithm is superior if n/r is large enough, a claim predicted by the complexity analysis in section 6.

Our algorithm is inferior to `bandr/tql2` and `tred2/tql2` with respect to accuracy. Table 7.2 shows that the results yielded by the divide and conquer algorithm lose 2 to 4 decimal digits compared with EISPACK. (In Table 7.2, δ_{ij} denotes the Kronecker symbol.) This is not surprising when we recall the difficulties that are encountered already in the tridiagonal divide and conquer algorithm. In their analysis, Sorensen and Tang [32] show that the secular equation which corresponds to the Weinstein determinant in our case has to be evaluated in double precision to guarantee full accuracy in the computed eigenvectors. Without this precaution, orthogonality among eigenvectors may be lost if close eigenvalues are present in the original or in one of the divided eigenvalue problem.

Problem size	Algorithm	$\max_j \|A\mathbf{q}_j - \lambda_j \mathbf{q}_j\|$	$\max_{i,j} \|\mathbf{q}_i^T \mathbf{q}_j - \delta_{ij}\|$
(100,3)	bandr/tql2	2.71E-14	1.04E-14
	tred2/tql2	4.67E-14	1.98E-14
	new	1.78E-10	3.41E-10
(100,6)	bandr/tql2	2.30E-14	7.77E-15
	tred2/tql2	3.08E-14	1.04E-14
	new	4.21E-12	3.67E-12
(200,6)	bandr/tql2	9.03E-14	1.64E-14
	tred2/tql2	8.76E-14	1.95E-14
	new	1.72E-11	4.08E-11
(300,6)	bandr/tql2	6.65E-13	2.11E-14
	tred2/tql2	1.28E-12	4.09E-14
	new	2.28E-09	4.42E-10

Table 7.2: Accuracy of computed eigenvalues and vectors

8. A modified algorithm

The motivation for the modification of the algorithm stemmed from the work of Gates [19], who changed the Dongarra-Sorensen divide and conquer algorithm in such a way that the eigenvectors are not computed by a formula corresponding to (4.11) but by inverse vector iteration. Orthogonality among the eigenvectors is obtained by reorthogonalizing those approximate eigenvectors corresponding to close eigenvalues. Gates' implementation of the tridiagonal divide and conquer method permits the deflation technique of the Dongarra-Sorensen algorithm. Thus, orthogonality among the eigenvectors is maintained without resorting to double precision computations while deflation can be taken over yielding good speedups over `tql2` even on sequential computers.

Our implementation finds those eigenvalues of A_0 which are also eigenvalues of A in advance in the fist for-loop of algorithm Assemble. We take this information into account in that the zerofinder doesn't have to find these roots again. But we do not deflate, i.e. reduce the problem size for the zerofinding step.

We introduced an additional synchronization point after step (ii) in the second for-loop in algorithm Assemble. At this point all the eigenvalues of the matrix A have been determined. After having sorted these eigenvalues we gather them in groups such that two different groups do not contain close eigenvalues. Closeness is defined as in the EISPACK routines `tinvit` or `bandv`: Two eigenvalues λ_i and λ_j are close if $|\lambda_i - \lambda_j| < \|A\|/1000$. As we know all eigenvalues of A, we use $\|A\|_2$ in this inequality. Possible reorthogonalization between eigenvectors corresponding to close eigenvalues can occur only within groups. Eigenvectors corresponding to different groups can therefore be computed in parallel. This selection of the groups may cause load imbalance. This did not happen in our test examples. But load imbalance is inevitable in connection with inverse vector iteration combined with reorthogonalization.

Inverse vector iteration has been implemented in a similar way as in subroutine

symray of Martin and Wilkinson [26]. The storage of the band matrix, Gaussian elimination with partial pivoting and solution of linear systems of equations with the band matrices are done in subroutines which are modifications of the LINPACK routines sgbfa and sgbsl [13, Chapter 2]. In particular, the shifted matrix $A - \sigma$ was stored with the whole band in a workspace of size $(3r + 1)n$. This workspace was necessary as the symmetry of the matrix gets lost in the process of Gaussian elimination with pivoting.

Inverse vector iteration was started with a back-substitution $U\mathbf{x} = \mathbf{e}$, where U is the upper-triangular factor of the Gaussian elimination of $A-\sigma$ and $\mathbf{e} = (1, \ldots, 1)^T$. Usually this back-substitution was sufficient to obtain convergence in the iteration. Implementing the inverse vector iteration this way gave accuracies comparable to the ones in Table 7.2. To further improve the accuracy of the computed eigenpairs we performed an additional iteration step to obtain a very precise eigenvector. The associated eigenvalue is then updated by the Rayleigh quotient of the improved eigenvector [26].

To avoid overflow in the neighborhood of the poles of $\det W(\lambda)$, we performed our calculations again with the function f in (7.1). But we used this representation of f only in a certain distance from the poles. Close to the poles we worked with extended Weinstein matrices as defined implicitly in (4.4). Close to the, say, upper endpoint of the interval $(\lambda_{j-1}^{(0)}, \lambda_j^{(0)})$ we represented f by

$$f(\lambda) = (\lambda_{j-1}^{(0)} - \lambda)^{\mu_{j-1}} \det W_e(\lambda),$$

where $W_e(\lambda)$ is the Weinstein matrix extended around $\lambda_j^{(0)}$

Problem size	Algorithm	$\max_j \|A\mathbf{q}_j - \lambda_j \mathbf{q}_j\|$	$\max_{i,j} \lvert\mathbf{q}_i^T \mathbf{q}_j - \delta_{ij}\rvert$
(100,3)	modified	2.66E-13	5.67E-14
(200,6)	modified	7.35E-13	2.92E-14
(300,6)	modified	1.50E-11	1.65E-12
(500,6)	modified	3.20E-12	3.75E-13

Table 8.1: Accuracy of computed eigenvalues and vectors for the modified algorithm

With the so modified algorithm we obtained the accuracies shown in Table 8.1. The results are still not as precise as those by the EISPACK methods. But the accuracy loss seems to be acceptable now.

Table 8.2 shows timings that we obtained with this modified divide and conquer algorithm. They show, that the price we have to pay for the increased accuracy was high!

Let us consider the complexity of the modified algorithm. For the complexity analysis of the modified algorithm. We assume that we have to perform one back-substitution and one full solve with a band matrix. We neglect the cost of possible reorthogonalizations. So we have to replace the term $\sum_{l=0}^{q-1} 2^l n_l [n_l^2 + n_l]$ in

Problem size	Number of processors	Elapsed times [seconds]			Speedups	
(n,r)	p	bandr/ tql2	tred2/ tql2	mod algorithm	mod vs. mod($p=1$)	mod vs. EISPACK
(100,3)	1	2.53	2.59	8.37	1.00	0.30
(100,3)	2			4.44	1.89	0.57
(100,3)	3			3.07	2.73	0.82
(100,3)	4			2.44	3.43	1.04
(100,3)	5			2.04	4.10	1.24
(100,3)	6			1.69	4.95	1.50
(200,6)	1	19.54	18.21	54.98	1.00	0.36
(200,6)	2			28.49	1.93	0.70
(200,6)	3			19.61	2.80	1.01
(200,6)	4			15.46	3.56	1.28
(200,6)	5			13.45	4.09	1.47
(200,6)	6			10.71	5.13	1.85
(300,6)	1	63.17	62.04	118.49	1.00	0.52
(300,6)	2			61.35	1.93	1.01
(300,6)	3			42.22	2.81	1.47
(300,6)	4			33.28	3.56	1.86
(300,6)	5			30.11	3.94	2.06
(300,6)	6			22.81	5.19	2.71
(500,6)	1	303.34	—	317.06	1.00	0.96
(500,6)	2			162.72	1.95	1.86
(500,6)	3			111.98	2.83	2.70
(500,6)	4			87.52	3.62	3.46
(500,6)	5			78.18	4.06	3.88
(500,6)	6			60.58	5.23	5.01

Table 8.2: Timings and speedups for modified algorithm

$Z_{\mathrm{d\&c}}^{(1\times r)}(n,r,p,s)$ by [13, Ch. 2]

$$\sum_{l=0}^{q-1} 2^l n_l [4n_l r^2 + 10 n_l r] = 2n^2 \frac{p-1}{p}(4r^2+10r). \tag{8.1}$$

This gives

$$Z_{\mathrm{d\&c,mod}}^{(1\times r)}(n,r,p,s) \cong \frac{35n^3}{3p^2} + \frac{n^2}{p}(2(p-1)((s+4)r^2+(3s+10)r+1)) \tag{8.2}$$

Notice that we have replaced an $O(n^3)$ term by an $O(n^2)$ term! However, the constant of the $O(n^2)$ term is large. Nevertheless, we can state that the modified algorithm is superior to our first as well as to bandr/tql2, if only the size of the problem at hand is sufficiently large.

Notice also, that the removed $O(n^3)$ term caused the large data transfer and thus the performance degradation of the divide and conquer algorithm on the hypercube [21]. Inverse vector iteration potentially reduces communication among

processors. Therefore, we expect our second algorithm to be well suited for implementation on a MIMD multiprocessor computer with distributed memory.

9. Concluding remarks

We have implemented two divide and conquer algorithms for computing the complete eigensolution of banded symmetric matrices. The proposed algorithms have serial complexities with smaller constants in the n^3 term than the `bandr/tql2` combination, the EISPACK solution path for the bandsymmetric eigenvalue problem. This means that our algorithms are faster than the EISPACK routines if the size of the problems are sufficiently large. Our numerical experiments on the Alliant FX-80 show that the break-even point is around a few hundred. The divide and conquer algorithms can be efficiently implemented on multiprocessor computers. The first proposed divide and conquer algorithm can be considered a generalization of the algorithm of Dongarra and Sorensen [14] for the tridiagonal eigenvalue problem. In contrast to this algorithm, the second proposed algorithm, a combination of divide and conquer and inverse vector iteration, computes the eigensolutions almost as accurate as the EISPACK routines. The first algorithm is faster than the second one. It is, as the Dongarra-Sorensen algorithm, suited for implementation on shared memory multiprocessors. The second algorithm may be suited for distributed memory MIMD computer as well. This issue is currently under investigation.

References

[1] P. Arbenz, Divide-and-conquer algorithms for the computation of the SVD of bidiagonal matrices, in: J. Dongarra, I. Duff, P. Gaffney, and S. McKee, eds., *Vector and Parallel Computing* (Ellis Horwood, Chichester, 1989) 1–10.

[2] P. Arbenz, Computing eigenvalues of banded symmetric Toeplitz matrices, *SIAM J. Sci. Stat. Comput.* **12** (1991) 743–754.

[3] P. Arbenz, W. Gander, and G. H. Golub, Restricted rank modification of the symmetric eigenvalue problem: Theoretical considerations, *Linear Algebra Appl.* **104** (1988) 75–95.

[4] P. Arbenz and G. H. Golub, On the spectral decomposition of Hermitian matrices modified by low rank perturbations with applications, *SIAM J. Matrix Anal. Appl.* **9** (1988) 40–58.

[5] Z. Bai and J. Demmel, On a block implementation of Hessenberg multishift QR iteration, *Int. J. High Speed Comput.* **1** (1989) 97–112.

[6] C. Beattie, An extension of Aronszajn's rule: Slicing the spectrum for intermediate problems, *SIAM J. Numer. Anal.* **24** (1987) 828–843.

[7] C. Beattie and D. Fox, Schur complements and the Weinstein–Aronszajn theory for modified matrix eigenvalue problems, *Linear Algebra Appl.* **108** (1988) 37–61.

[8] C. A. Beattie and C. J. Ribbens, Parallel solution of a generalized eigenvalue problem, in: D. C. Sorensen, ed., *Proceedings of the Fifth SIAM Conference on Parallel Processing* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1991).

[9] R. P. Brent, *Algorithms for Minimization without Derivatives* (Prentice-Hall, Englewood Cliffs, NJ, 1973).

[10] J. R. Bunch, C. P. Nielson, and D. C. Sorensen, Rank–one modification of the symmetric eigenproblem, *Numer. Math.* **31** (1978) 31–48.

[11] S. L. Campbell and C. D. Meyer, *Generalized Inverses of Linear Transformations* (Pitman, London, 1979).

[12] J. J. M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.* **36** (1981) 177–195.

[13] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users Guide* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1979).

[14] J. J. Dongarra and D. C. Sorensen, A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM J. Sci. Stat. Comput.* **8** (1987) s139–s154.

[15] J. J. Dongarra and D. C. Sorensen, A portable environment for developing parallel FORTRAN programs, *Parallel Computing* **5** (1987) 175–186.

[16] J. J. Dongarra, D. C. Sorensen, K. Connolly, and J. Patterson, Programming methodology and performance issues for advanced computer architectures, *Parallel Computing* **8** (1988) 41–58.

[17] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling, Block reduction of matrices to condensed forms for eigenvalue computations, *J. Comput. Appl. Math.* **27** (1989) 215–227.

[18] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide Extension*, *Lecture Notes in Computer Science* 51 (Springer-Verlag, Berlin, 1977).

[19] K. Gates, Using inverse iteration to improve the divide and conquer algorithm, Technical Report 159, ETH Zürich, Departement Informatik, May 1991.

[20] G. H. Golub and C. F. van Loan, *Matrix Computations*, 2nd ed. (The Johns Hopkins University Press, Baltimore, MD, 1989).

[21] I. C. F. Ipsen and E. R. Jessup, Solving the tridiagonal eigenvalue problem on the hypercube, *SIAM J. Sci. Stat. Comput.* **11** (1990) 203–229.

[22] E. R. Jessup, A case against a divide and conquer approach to the nonsymmetric eigenvalue problem, Tech. Report (in preparation), Department of Computer Science, University of Colorado, Boulder, CO, 1991.

[23] L. Kaufman, Banded eigenvalue solvers on vector machines, *ACM Trans. Math. Softw.* **10** (1984) 73–86.

[24] S.-S. Lo, B. Philippe, and A. Sameh, A multiprocessor algorithm for the symmetric tridiagonal eigenvalue problem, *SIAM J. Sci. Stat. Comput.* **8** (1987) s155–s165.

[25] S. C. Ma, M. L. Patrick, and D. B. Szyld, A parallel, hybrid algorithm for the generalized eigenproblem, in: G. Rodrigue, ed., *Parallel Processing for Scientific Computing* (Society for Industrial and Applied Mathematics, Philadelphia, PA, 1989) 82–86.

[26] R. S. Martin and J. H. Wilkinson, Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices, *Numer. Math.* **9** (1967) 279–301. Reprinted in [34], pp. 70–92.

[27] D. O'Leary and G. W. Stewart, Computing the eigenvalues and eigenvectors of symmetric arrowhead matrices, *J. Comp. Phys.* **90** (1990) 497–505.

[28] B. N. Parlett, *The Symmetric Eigenvalue Problem* (Prentice Hall, Englewood Cliffs, NJ, 1980).

[29] H. R. Schwarz, Tridiagonalization of a symmetric band matrix, *Numer. Math.* **12** (1968) 231–241. Reprinted in [34], pp. 273–283.

[30] N. S. Sehmi, *Large Order Structural Eigenanalysis Techniques* (Ellis Horwood, Chichester, 1989).

[31] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide*, *Lecture Notes in Computer Science* 6, 2nd ed. (Springer-Verlag, Berlin, 1976).

[32] D. C. Sorensen and P. T. P. Tang, On the orthogonality of eigenvectors computed by divide-and-conquer techniques, Preprint MCS-P152-0490, Argonne National Laboratory, Mathematics and Computer Science Division, May 1990.

[33] A. Weinstein and W. Stenger, *Methods of Intermediate Problems for Eigenvalues* (Academic Press, New York, 1972).

[34] J. H. Wilkinson and C. Reinsch, eds., *Linear Algebra* (Springer-Verlag, Berlin, 1971).