An Algorithm and Architecture Based on Orthonormal μ–Rotations for Computing the Symmetric EVD

JÜRGEN GÖTZE [1]
Department of Electrical & Computer Engineering
Rice University
Houston, TX 77251–1892, U.S.A.
jugo@ece.rice.edu

GERBEN J. HEKSTRA
Department of Electrical Engineering
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
gerben@dutentb.et.tudelft.nl

Abstract

In this paper an algorithm and architecture for computing the eigenvalue decomposition (EVD) of a symmetric matrix is presented. The EVD is computed using a Jacobi-type method, where the angle of the rotations is approximated by an angle α_k, corresponding to an orthonormal μ–rotation. These orthonormal μ–rotations are based on the idea of CORDIC and share the property that performing the rotation requires a minimal number of shift–add operations. We present various methods of construction for such orthonormal μ–rotations of increasing complexity. Moreover, the computations to determine which angle α_k to use in the approximation of the optimal angle, can itself be expressed purely in orthonormal μ–rotations on the matrix data. The complexity of the used type of orthonormal μ–rotation decreases during the diagonalization of the matrix. A significant reduction of the number of required shift–add operations is achieved.

All types of fast, orthonormal μ–rotations (and the computation to determine the optimal angle) can be implemented as a cascade of only four basic types of shift–add stages. These stages can be executed on a modified sequential floating-point CORDIC architecture, making the presented algorithm highly suited for VLSI–implementation.

Keywords. Eigenvalue decomposition (EVD), Jacobi method, approximate rotations, orthonormal μ–rotations, floating-point CORDIC architecture.

[1] This work was supported by the Delft University of Technology, the Alexander von Humboldt foundation and Texas–ATP

1 INTRODUCTION

The EVD of a $n \times n$ symmetric matrix $\boldsymbol{A}$ is defined as

$$\boldsymbol{A} = \boldsymbol{Q}^T \boldsymbol{\Lambda} \boldsymbol{Q} \, ,$$

where $\boldsymbol{Q}$ is an orthogonal matrix ($\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$) and $\boldsymbol{\Lambda} = \mathrm{diag}[\lambda_1, \ldots, \lambda_n]$ is a diagonal matrix containing the eigenvalues λ_i of $\boldsymbol{A}$. Computing the EVD of a symmetric matrix is a frequently encountered problem in a great number of scientific applications, e.g. signal processing [1, 2, 3]. In order to meet the real time requirements present in many of these applications, it is often necessary to compute the EVD on a parallel architecture.

The method of choice for the fast parallel computation of the EVD is the Jacobi method, since it offers a significantly higher degree of parallelism than the respective QR–method [4, 5]. One sweep of a cyclic Jacobi method (consisting of $n(n \Leftrightarrow 1)/2$ rotation evaluations and their application to the matrix) can be implemented on an upper triangular array of processors with nearest neighbour interconnections in $O(n)$ time [4, 6]. Usually, $O(\log_2 n)$ sweeps are required [4]. Besides this suitability for a parallel implementation the Jacobi method offers a higher numerical accuracy than the QR–method [7].

The complexity of parallel or sequential implementations is mainly determined by the complexity of the rotation evaluation (angle computation) and application. Therefore, different strategies for modifying the rotations have been presented:

- **M1:** approximate rotations [6, 8],
- **M2:** factorized rotations [9, 10, 11] ,
- **M3:** CORDIC [5, 12],
- **M4:** Combination of the modifications M1–M3:
  - **M1+M2:** factorized approximate rotations [13, 14]
  - **M1+M3:** CORDIC–like approximate rotations (orthonormal μ–rotations) [15, 14]

Which of these modified schemes yields the most efficient parallel implementation depends on the particular parallel architecture. With respect to an VLSI–architecture a CORDIC–like method is favourable, whereby using CORDIC–like approximate rotations is advantageous compared to the use of the original (exact) CORDIC [15].

In this paper an algorithm for computing the EVD of a symmetric matrix by a Jacobi–type method is presented. The algorithm employs different types of simple orthonormal μ–rotations. These orthonormal μ–rotations realize approximate rotations (i.e. the off–diagonal element is only reduced instead of annihilated as by an exact rotation). The complexity and cost (number of required shift–add operations) of the different orthonormal μ–rotations decreases as the rotation angle decreases. Since the required rotation angle decreases during the course of the Jacobi method, the type of orthonormal μ–rotation used during the course of the algorithm becomes less complex, i.e., the type of orthonormal μ–rotation is adapted to the stage of the diagonalization. Furthermore, it is also possible to adapt the accuracy of the rotation by increasing the number of orthonormal μ–rotations used in the approximation of the angle in order to regain the quadratic convergence of the exact Jacobi method [15, 16]. The presented algorithm is highly suited for a dedicated (parallel) VLSI architecture. For that purpose an elementary CORDIC–like architecture is presented. The different types of orthonormal μ–rotations can be realized by a cascade of these elementary building blocks.

In section 2 we review the cyclic Jacobi algorithm for computing the EVD of a symmetric matrix and describe the idea of using approximate rotations. Section 3 gives the basic definition

of fast, orthonormal μ–rotations and presents several methods for unscaled orthonormal μ–rotations (method I, II, III), or for μ–rotations (method IV) requiring extra scaling iterations to make them orthonormal. The methods I–III are obtained by truncating the Taylor expansions of the sine–cosine with varying accuracy. Method IV is derived from the double rotation method presented in [17]. It is also shown how to construct a set $\mathcal{A}$ of approximation angles using the above methods, and how to select the optimal approximation angle from this set. The optimal approximate angle is the one which effects in the greatest reduction of the off-diagonal matrix entry. The computation for selecting the optimal approximate angle can also be expressed in terms of μ–rotations. In section 4 it is demonstrated how the used type of orthonormal μ–rotation is automatically adapted during the course of the Jacobi method. The resulting reduction of the required number of shift–add operations per sweep is shown. Furthermore, in order to reduce the off–chip communication overhead the convergence speed of the algorithm can be steered by adaptively varying the number of μ–rotations used in the approximation of the rotation (i.e. the accuracy of the approximation). Section 5 presents a floating point architecture which comprises four types of elementary shift–add operations. All orthonormal μ–rotations are realized as a cascade of such elementary operations, executed in sequence on this architecture. Section 6 concludes the paper.

2 JACOBI'S METHOD

With respect to a regular (sequential or parallel) implementation of Jacobi's method for computing the EVD of a symmetric matrix a cyclic Jacobi method is used [4].

2.1 Cyclic Jacobi method

The cyclic–by–row Jacobi method computes the EVD of a $n \times n$ symmetric matrix by applying a sequence of orthonormal rotations to the left and right of $\boldsymbol{A}$, as shown in the following algorithm:

$h := 1$; $\boldsymbol{A}^{(1)} := \boldsymbol{A}$
for $s := 1$ **to** number of sweeps
  for $p := 1$ **to** $n \Leftrightarrow 1$
    for $q := p + 1$ **to** n
      $\boldsymbol{A}^{(h+1)} := \boldsymbol{J}_{pq}(\theta_h)\boldsymbol{A}^{(h)}\boldsymbol{J}_{pq}^{T}(\theta_h)$
      $h := h + 1$
    end
  end
end

where $\boldsymbol{J}_{pq}(\theta)$ is an orthonormal plane rotation over the angle θ in the (p, q) plane, and defined by $(\cos\theta, \Leftrightarrow\sin\theta, \sin\theta, \cos\theta)$ in the (pp, pq, qp, qq) positions of the $n \times n$ identity matrix.

The index pairs are choosen in the cyclic–by–row manner

$$(p, q) = (1,2)(1,3)\ldots(1,n)(2,3)\ldots(2,n)\ldots(n \Leftrightarrow 1, n)\,. \tag{1}$$

The execution of all $N = n(n \Leftrightarrow 1)/2$ index pairs (p, q) according to (1) is called a sweep. Since the matrices $\boldsymbol{A}^{(h)}$ remain symmetric for all h, it is sufficient to work with the upper triangular part of the matrix throughout the algorithm.

Defining the off–diagonal quantity $S^{(h)}$ by

$$S^{(h)} = \sqrt{\frac{1}{2}\left[\left\|\boldsymbol{A}^{(h)}\right\|_F^2 \Leftrightarrow \sum_{i=1}^{n}\left(a_{ii}^{(h)}\right)^2\right]}\,, \tag{2}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, the execution of a similarity transformation $\boldsymbol{A}^{(h+1)} = \boldsymbol{J}_{pq}(\theta_h)\boldsymbol{A}^{(h)}\boldsymbol{J}_{pq}^T(\theta_h)$ yields [18]:

$$\left[S^{(h+1)}\right]^2 = \left[S^{(h)}\right]^2 \Leftrightarrow \left[\left(a_{pq}^{(h)}\right)^2 \Leftrightarrow \left(a_{pq}^{(h+1)}\right)^2\right]\,. \tag{3}$$

Obviously, a reduction of $S^{(h)}$ is obtained if $|a_{pq}^{(h+1)}| < |a_{pq}^{(h)}|$. This reduction is maximal if $a_{pq}^{(h+1)} = 0$. Therefore,

$$\lim_{h\to\infty} S^{(h)} \to 0 \qquad \Leftrightarrow \qquad \lim_{h\to\infty} \boldsymbol{A}^{(h)} \to \mathrm{diag}[\lambda_1, \ldots, \lambda_n]\,. \tag{4}$$

Without loss of generality we will drop the index h and only consider the 2×2 symmetric EVD subproblem

$$\begin{bmatrix} a'_{pp} & a'_{pq} \\ a'_{pq} & a'_{qq} \end{bmatrix} = \begin{bmatrix} \cos\theta & \Leftrightarrow\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a_{pp} & a_{pq} \\ a_{pq} & a_{qq} \end{bmatrix} \begin{bmatrix} \cos\theta & \Leftrightarrow\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}^T \tag{5}$$

in the sequel. Thus, a'_{pq} is given by

$$a'_{pq} = \frac{1}{2}\left[(2a_{pq})\cos 2\theta \Leftrightarrow (a_{qq} \Leftrightarrow a_{pp})\sin 2\theta\right]\,. \tag{6}$$

Solving $a'_{pq} = 0$ one obtains the optimal angle of rotation θ_{opt}, for which maximal reduction of S (i.e. $a'_{pq} = 0$) is achieved:

$$\theta_{opt} = \frac{1}{2}\arctan(\tau)\,, \tag{7}$$

where $\tau = \frac{2a_{pq}}{a_{qq}-a_{pp}}$, and where the range of θ_{opt} is limited to $|\theta_{opt}| \leq \frac{\pi}{4}$.

2.2 Approximate rotations

Using approximate rotations enables the reduction of the complexity of the vectorization and the rotation mode [6, 8, 13]. For a reduction of the off–diagonal quantity S it is not necessary to meet $a'_{pq} = 0$, but it is sufficient, when using an approximate angle θ_{app}, that

$$\left|a'_{pq}\right| = |d| \cdot |a_{pq}| \;\text{ with } 0 \leq |d| < 1\,, \tag{8}$$

where the reduction d is a function of the approximate angle θ_{app} and the matrix data, given by:

$$d(\theta_{app}, \tau) = \cos 2\theta_{app} \Leftrightarrow \frac{1}{\tau}\sin 2\theta_{app}\,. \tag{9}$$

This is the basis for the design of approximate rotations, i.e. rotations that meet (8).

The approach used is to construct a set $\mathcal{F} = \{\boldsymbol{F}_0, \boldsymbol{F}_{-1}, \ldots\}$ of fast orthonormal μ–rotations, where $\boldsymbol{F}_k$ implements the rotation over an angle α_k in a minimal number of shift–add operations.

The angles α_k form the ordered set of approximation angles $\mathcal{A} = \{\alpha_0, \alpha_{-1}, \ldots\}$. Both angle α_k and orthonormal μ–rotation $\boldsymbol{F}_k$ are selected by the integer angle index k, satisfying $k \leq 0$. In section 3, we will present methods to construct such fast orthonormal μ–rotations and how to determine the optimal angle index k.

Using approximate rotations must be paid by an increase in the number of required sweeps. It has been shown in [8, 13, 15], however, that the overall cost, in terms of shift–add operations, obtained for the Jacobi method using approximate rotations is significantly lower than for the Jacobi method using exact rotations.

3 METHODS FOR ORTHONORMAL ROTATIONS

We define the close-to-orthonormal μ–rotation $\boldsymbol{F}$ as given by the 2×2 rotation matrix

$$\boldsymbol{F} = \begin{bmatrix} \hat{c} & \Leftrightarrow\sigma\hat{s} \\ \sigma\hat{s} & \hat{c} \end{bmatrix} = \hat{m} \cdot \boldsymbol{J}(\theta) \tag{10}$$

with $\hat{m}$ being the scaling factor of the matrix, given by

$$\hat{m} = \sqrt{\hat{c}^2 + \hat{s}^2} = 1 + \varepsilon\,. \tag{11}$$

The $\hat{c}, \hat{s}$ are pairwise approximations of a sine/cosine pair, satisfying $0 \leq \hat{c}, \hat{s} \leq 1$, and chosen such that

1. the multiplication with $\hat{c}$ and $\hat{s}$ is cheap to compute, or that the combined evaluation of the rotation has a cheap implementation, basically only a small number of shift–add operations.

2. the error in scaling ε is smaller than the required accuracy. When this is the case, the effect of the error in the scaling (orthonormality) is overshadowed by the rounding error in the computation. Hence the term "close-to-orthonormal" applies for this type of rotations.

The angle of rotation $\theta = \sigma \cdot \alpha$ is determined by the direction of rotation $\sigma \in \{\Leftrightarrow 1, +1\}$, signifying clockwise or counterclockwise rotation, and by the absolute angle of rotation α, with $\alpha < \pi/2$, which is fixed through the choice of the $\hat{c}, \hat{s}$ pair as:

$$\alpha = \arctan(\frac{\hat{s}}{\hat{c}}) \tag{12}$$

We have at our disposal a number of methods to systematically arrive at $\hat{c}, \hat{s}$ pairs of varying accuracy and for varying angles of rotation (= varying angle index k). We will present two classes of methods for orthonormal μ–rotations, and select four methods from these, taking into account the cost involved for performing such rotations.

3.1 Unscaled orthonormal μ–rotations

The simplest of the methods is the method I rotation, which is the same as the CORDIC μ–rotation, with

$$\begin{aligned} \hat{c} &= 1 \\ \hat{s} &= 2^k \\ \hat{m} &= (1 + 2^{2k})^{\frac{1}{2}} \end{aligned} \tag{13}$$

where k, with $k \leq 0$ and integer, is the angle index. The smaller the value of k, the smaller the angle, and the more the scaling factor $\hat{m}$ approaches unity.

For (floating-point) computation, with the mantissa size n_m, the maximum round-off error is equal to $\frac{1}{2}2^{-n_m} = 2^{-(n_m+1)}$.

Hence we define the working limit G_I for the angle index such that if $k \leq G_I$ then $\hat{m}$ satisfies:

$$1 \Leftrightarrow 2^{-(n_m+1)} < \hat{m} < 1 + 2^{-(n_m+1)} \tag{14}$$

and the scaling effect of the rotation can be neglected. Solving (14) for the working limit G_I by substituting the actual value of $\hat{m}$ of equation (13) in (14) yields:

$$G_I = \left\lfloor \frac{\Leftrightarrow n_m}{2} \right\rfloor \tag{15}$$

For example, for $n_m = 32$, we can use this method I for $k = \Leftrightarrow 16, \ldots, \Leftrightarrow 32$ without scaling.

The more accurate method II rotations are given by

$$\begin{array}{rcl} \hat{c} & = & 1 \Leftrightarrow 2^{2k-1} \\ \hat{s} & = & 2^k \\ \hat{m} & = & (1 + 2^{4k-2})^{\frac{1}{2}} \end{array} \tag{16}$$

while the even more accurate method III rotations are given by

$$\begin{array}{rcl} \hat{c} & = & 1 \Leftrightarrow 2^{2k-1} \\ \hat{s} & = & 2^k \Leftrightarrow 2^{3k-3} \\ \hat{m} & = & (1 + 2^{6k-6})^{\frac{1}{2}} \end{array} \tag{17}$$

These methods have working limits G_{II} and G_{III}, respectively, derived in a similar manner to G_I:

$$\begin{array}{rcl} G_{II} & = & \left\lfloor \frac{\Leftrightarrow n_m + 2}{4} \right\rfloor \\ G_{III} & = & \left\lfloor \frac{\Leftrightarrow n_m + 6}{6} \right\rfloor , \end{array} \tag{18}$$

i.e., for $n_m = 32$ we can use method II for $k \leq 8$ and method III for $k < 4$ without scaling.

As each of the previous three methods is limited to a certain working limit, new methods must be developed for more and more accurate rotations. An alternative is to use a fixed method of given accuracy and apply additional scaling iterations to achieve the required accuracy.

3.2 Orthonormal μ–rotations with extra scaling

The rotation used is a double rotation of method I, rotating twice over an angle defined by the angle index of $k \Leftrightarrow 1$, resulting in:

$$\begin{array}{rcllcl} \hat{c} & = & (1)^2 \Leftrightarrow (2^{k-1})^2 & = & 1 \Leftrightarrow 2^{2k-2} \\ \hat{s} & = & 2(2^{k-1}) & = & 2^k \\ \hat{m} & = & \sqrt{1 + 2^{2(k-1)}}^2 & = & 1 + 2^{2(k-1)} \end{array} \tag{19}$$

Note that the scaling is no longer a square root function. We exploit this fact in the following fast converging scaling sequence.

We define the additional scaling $K_m = \prod_{i=0}^{m} \kappa_i$, where $m \geq 0$ is the number of scaling steps κ_i. The scaling steps are given by

$$\begin{array}{rcll} \kappa_0 & = & 1 & \\ \kappa_1 & = & (1 \Leftrightarrow 2^{2(k-1)}) & \\ \kappa_i & = & (1 + 2^{2^i(k-1)}) & \text{for } i \geq 2 \end{array} \tag{20}$$

Hence the method *IV* scaled orthonormal μ–rotation is given by $\hat{c}$, $\hat{s}$ of (19) and the scaling procedure according to (20). Therefore, we have $K_m \boldsymbol{F} = K_m \hat{m} \boldsymbol{J}(\theta)$ and the number of scaling steps m must be determined such that $1 \Leftrightarrow 2^{-(n_m+1)} < K_m \hat{m} < 1 + 2^{-(n_m+1)}$, i.e. the scaling effect can again be neglected for the mantissa size n_m. With $\hat{m}$ of (19) and the scaling recursion (20) one obtains

$$\begin{array}{rcl} \hat{m} K_m & = & (1 + 2^{2(k-1)})(1 \Leftrightarrow 2^{2(k-1)})(1 + 2^{4(k-1)})(1 + 2^{8(k-1)}) \cdots (1 + 2^{2^m(k-1)}) \\ & = & 1 \Leftrightarrow 2^{2^{(m+1)}(k-1)} \end{array} \tag{21}$$

As for the unscaled methods, we can now derive the working limit G_{IV}, which is a function of both the accuracy n_m and the number of scaling steps m, as being:

$$G_{IV} = \left\lceil \frac{\Leftrightarrow(n_m + 1)}{2^{(m+1)}} \right\rceil \tag{22}$$

In a similar way, we can compute the dual form, which is the limit M, being the minimum number of scaling steps m necessary to reach the required accuracy. Without proof, we state that the limit M is given by

$$M = \left\lfloor \log_2 \left(\frac{\Leftrightarrow(n_m + 1)}{k \Leftrightarrow 1} \right) \right\rfloor \tag{23}$$

Using this rotation method, and taking the minimum number of scaling steps necessary, as dictated by M, we can provide orthonormal μ–rotations for rotating over large angles and for high accuracy. Note that the rotation itself costs 4 shift–add operations, or two *pairs* of shift–add operations, while the scaling costs M *pairs* of shift–add operations. In general, we can say that the cost of method *IV* is given by $2 \cdot (2 + M)$ shift–add operations.

For example: for $n_m = 32$, we can construct the rotation for $k = \Leftrightarrow 2$, with the minimum number of scaling steps, $m = M = 3$, as

$$\boldsymbol{F}_{-2}(\sigma) = (1 \Leftrightarrow 2^{-6})(1 + 2^{-12})(1 + 2^{-24}) \begin{bmatrix} 1 \Leftrightarrow 2^{-6} & \Leftrightarrow \sigma(2^{-2}) \\ \sigma(2^{-2}) & 1 \Leftrightarrow 2^{-6} \end{bmatrix}. \tag{24}$$

3.3 Constructing the set of μ–rotations

The ordered set of approximation angles $\mathcal{A}$, for a given accuracy n_m, is constructed using the aforementioned methods. For $n_m = 32$ table 1 shows when to select which method, depending on the value of the angle index k, as well as the angle, and the cost for rotation and possible scaling. The orthonormal μ–rotations are chosen such that they satisfy the accuracy condition (14) using the cheapest possible method.

angle index	method	angle	cost (shift-add operations)	
k		α_k	rot.	scl.
0	IV	0.92730	4	10
-1	IV	0.48996	4	8
-2	IV	0.24871	4	6
-3	IV	0.12484	4	6
-4	IV	$6.24797\,10^{-2}$	4	4
-5	III	$3.12513\,10^{-2}$	6	0
-6	III	$1.56252\,10^{-2}$	6	0
-7	III	$7.81252\,10^{-3}$	6	0
-8	II	$3.90626\,10^{-3}$	4	0
-9	II	$1.95313\,10^{-3}$	4	0
-10	II	$9.76563\,10^{-4}$	4	0
-11	II	$4.88281\,10^{-4}$	4	0
-12	II	$2.44141\,10^{-4}$	4	0
-13	II	$1.22070\,10^{-4}$	4	0
-14	II	$6.10352\,10^{-5}$	4	0
-15	II	$3.05176\,10^{-5}$	4	0
-16	I	$1.52588\,10^{-5}$	2	0
-17	I	$7.62939\,10^{-6}$	2	0
-18	I	$3.81470\,10^{-6}$	2	0
-19	I	$1.90735\,10^{-6}$	2	0
-20	I	$9.53674\,10^{-7}$	2	0
-21	I	$4.76837\,10^{-7}$	2	0
-22	I	$2.38419\,10^{-7}$	2	0
-23	I	$1.19209\,10^{-7}$	2	0
-24	I	$5.96046\,10^{-8}$	2	0
-25	I	$2.98023\,10^{-8}$	2	0
-26	I	$1.49012\,10^{-8}$	2	0
-27	I	$7.45058\,10^{-9}$	2	0
-28	I	$3.72529\,10^{-9}$	2	0
-29	I	$1.86265\,10^{-9}$	2	0
-30	I	$9.31323\,10^{-10}$	2	0
-31	I	$4.65661\,10^{-10}$	2	0
-32	I	$2.32831\,10^{-10}$	2	0

Table 1: The set $\mathcal{A}$ of μ–rotations for 32-bit accuracy, showing the method used, the angle and the cost of rotation and scaling.

3.4 Determining the optimal approximate rotation

The crucial point for approximate rotations is to find the approximate angle $\theta_{app} = \sigma \cdot \alpha_k$, where $\sigma \in \{\Leftrightarrow 1, +1\}$ is the direction of the rotation and the angle $\alpha_k \in \mathcal{A}$, with the angle index k, is chosen such that $|d(\theta_{app}, \tau)|$ is minimal.

The direction of rotation σ follows from the sign of the optimal angle θ_{opt}, and is given by:

$$\sigma = \text{sign}(\tau) = \text{sign}(a_{pq}) \cdot \text{sign}(a_{qq} \Leftrightarrow a_{pp}) \tag{25}$$

In order to determine the angle index k, we introduce the working domain limits g_k, and define the condition for choosing the angle α_k as being

$$g_{k-1} < |\tau| \leq g_k \tag{26}$$

where g_k is determined such that, when τ satisfies (26), then $|d(\sigma \cdot \alpha_k, \tau)|$ is minimal over the set of angles $\mathcal{A}$. Note, that minimizing $|d(\sigma \cdot \alpha_k, \tau)|$ is equivalent to choosing the angle $\alpha_k (= \theta_{app})$ which is closest to the optimal angle θ_{opt}. The limit g_k follows from the solution of

$$d(\alpha_k, g_k) = \Leftrightarrow d(\alpha_{k+1}, g_k)\,, \tag{27}$$

i.e. the point in the domain $|\tau|$ where α_k and α_{k+1} lead to the same reduction of the off–diagonal element. The solution of (27) yields

$$g_k = \tan(\alpha_k + \alpha_{k+1}) = \tan \gamma_k\,, \tag{28}$$

where $\gamma_k = \alpha_k + \alpha_{k+1}$ is the working limit in the angle domain.

In [15], for this particular set of approximative angles, and through the use of floating-point arithmetic the search is narrowed down to check which of three consecutive angles gives the maximal reduction (minimal $|d|$). Here we show that this selection can also be executed by using μ–rotations.

Construct the vector v given by

$$\boldsymbol{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} a_{qq} \Leftrightarrow a_{pp} \\ 2a_{pq} \end{bmatrix}\,. \tag{29}$$

The angle between v and the x-axis is equal to the required $\arctan(\tau) = 2\theta_{opt}$. To test whether $|\tau|$ is greater or smaller than the limit g_k is equivalent to checking whether this angle is resp. greater or smaller in absolute value to the limit angle γ_k.

Hence we rotate $\boldsymbol{v}$ over the angle $\Leftrightarrow\sigma \cdot \gamma_k = \Leftrightarrow\sigma \cdot (\alpha_k + \alpha_{k+1})$, to obtain $\boldsymbol{v}' = [v'_x\, v'_y]^T$ (see Figure 1). When the y-component v'_y is of the *same* sign as v_y, then the angle between v and the x-axis is larger than γ_k, and we will select α_{k+1} as the approximate angle. If, however, v'_y is of *opposite* sign to v_y, then the angle is smaller, and we select α_k. The rotation over $\Leftrightarrow\sigma \cdot \gamma_k$ is performed as two consecutive rotations over α_k and α_{k+1}, implemented as fast orthonormal μ–rotations

$$\boldsymbol{v}' = \boldsymbol{J}(\Leftrightarrow\sigma\gamma_k) \cdot \boldsymbol{v} = \boldsymbol{F}_{k+1}(\Leftrightarrow\sigma) \cdot \boldsymbol{F}_k(\Leftrightarrow\sigma) \cdot \boldsymbol{v} \tag{30}$$

Obviously, for our example in Figure 1 we have to use α_k since $v_y > 0$ and $v'_y < 0$ holds.

The search for the optimal angle $\alpha_k \in \mathcal{A}$, which effects in the greatest reduction of the off–diagonal element a_{pq}, can be narrowed to a check of three consecutive angles. This can simply

be achieved by looking at the exponents of $[v_x, v_y]$. We obtain an estimate for the optimal angle index k_{opt} as follows: $k = \exp(v_y) \Leftrightarrow \exp(v_x)$ (if $k > 0$ set $k = 0$), where $\exp(f)$ denotes the exponent of the floating point number f. Based on this estimate the optimal angle index $k_{opt} \in \{k \Leftrightarrow 1, k, k+1\}$ can be determined. This is illustrated in Figure 2. The domains of possible values for v_x and v_y are indicated by black bars on the v_x and v_y axis, respectively. This domain is determined by the mantissas ($1/2 \leq \mathrm{man}(v_x) < 1$, $1/2 \leq \mathrm{man}(v_y) < 1$) which were not taken into consideration by determing the estimate k for the optimal angle index k_{opt}. These domains of the matissa describe a rectangle in the (v_x, v_y)-plane. It is easy to show that this rectangle covers at most three possible rotation angles, i.e., $\theta_{app} \in \{\alpha_{k-1}, \alpha_k, \alpha_{k+1}\}$.

Now the method described above (Figure 1) can be adapted to test which one of the three consecutive angles to use. For this we compute $\boldsymbol{v}'$ and $\boldsymbol{v}''$ according to

$$\begin{array}{rclcl} \boldsymbol{v}' & = & \boldsymbol{J}(\Leftrightarrow\sigma\gamma_k)\cdot\boldsymbol{v} & = & \boldsymbol{F}_{k+1}(\Leftrightarrow\sigma)\cdot\boldsymbol{F}_k(\Leftrightarrow\sigma)\cdot\boldsymbol{v} \\ \boldsymbol{v}'' & = & \boldsymbol{J}(\Leftrightarrow\sigma\gamma_{k-1})\cdot\boldsymbol{v} & = & \boldsymbol{F}_{k-1}(\Leftrightarrow\sigma)\cdot\boldsymbol{F}_k(\Leftrightarrow\sigma)\cdot\boldsymbol{v} \end{array} \tag{31}$$

and choose the correct angle of rotation, using the selection tree, based on the resulting signs of v'_y, v''_y

$$\begin{array}{ll} \alpha_{k+1} & \text{if } \mathrm{sign}(v_y) = \mathrm{sign}(v'_y) \\ \alpha_{k-1} & \text{if } \mathrm{sign}(v_y) \neq \mathrm{sign}(v''_y) \\ \alpha_k & \text{otherwise} \end{array} \tag{32}$$

If the intermediate results of the rotations are re-used, this selection mechanism costs at most three orthonormal μ–rotations (note that this selection can also be executed by unscaled μ–rotations). For our example in Figure 2 we obtain $v_y > 0$, $v'_y < 0$, $v''_y > 0$, such that α_k is the optimal approximate angle θ_{app}.

The resulting total reduction $|d(\tau, \theta_{app})|$, using the above method for determining the approximate angle is shown in figure 3 for the set of angles $\mathcal{A}$ with 32-bit accuracy.

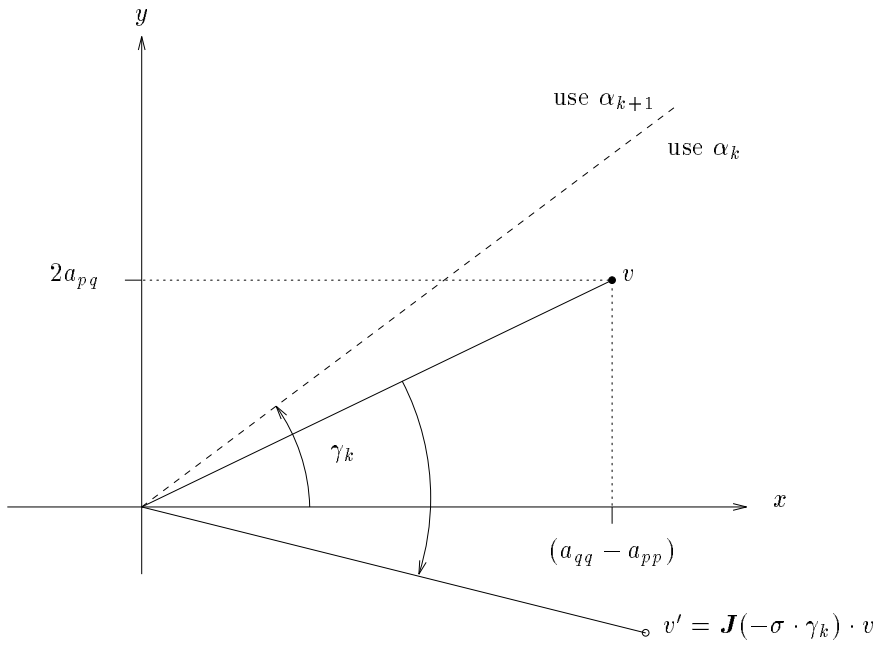

Figure 1: Seperation of consecutive angles α_k and α_{k+1}.

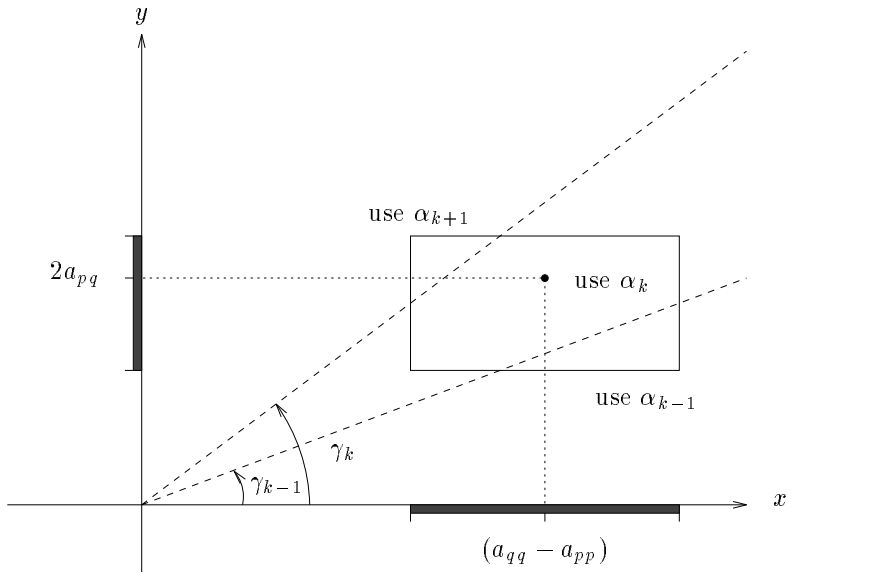

Figure 2: Determination of optimal approximate angle.

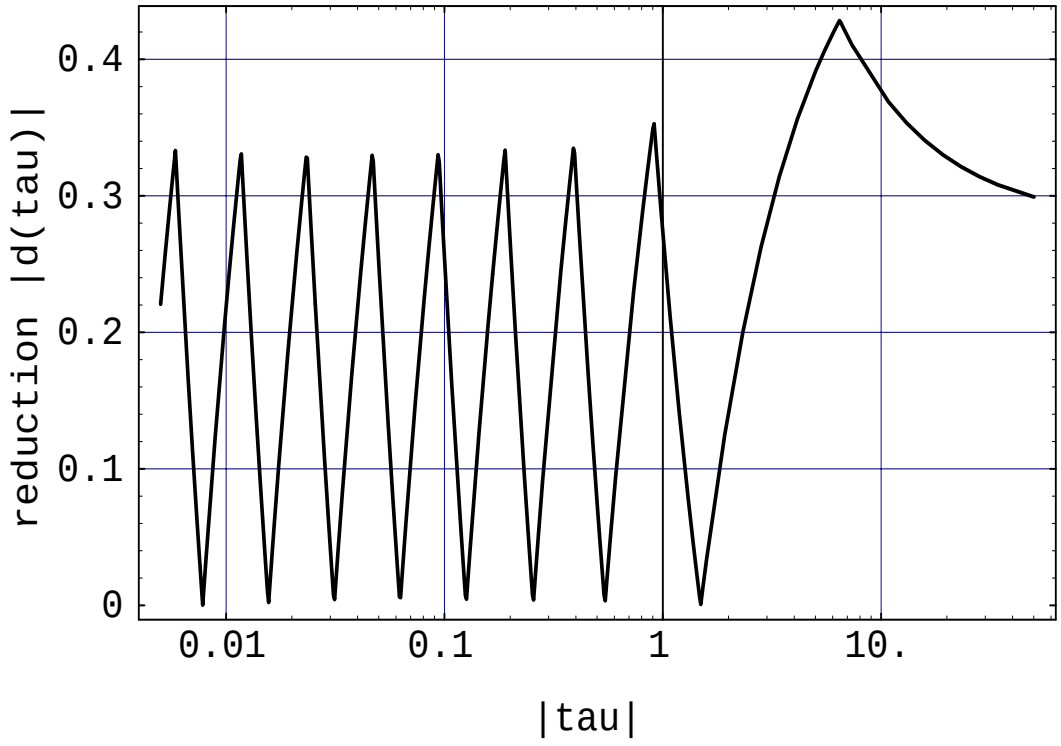

Figure 3: The total reduction as a function of $|\tau|$, for the set of angles with 32-bit accuracy

4 PERFORMANCE OF THE ALGORITHM

A random symmetric matrix $\boldsymbol{A}$ of dimension 20×20 is used to illustrate the performance of the algorithm. We assume a wordlength of 32 bit for all our implementations. The Jacobi method is terminated if the off–diagonal norm is smaller than $10^{-8} \cdot \|\boldsymbol{A}\|_F$.

In Figure 4 the reduction of the off–diagonal norm vs. the sweeps is shown. Obviously, using only one orthonormal μ–rotation for approximating the rotation shows no ultimate quadratic convergence (solid line) as does the Jacobi method with exact rotations (dash–dotted line). As shown in [15] the quadratic convergence of the Jacobi method can be regained by adapting (increasing) the number of orthonormal μ–rotations executed per plane rotation $\boldsymbol{J}_{pq}$. Executing $r \geq 1$ orthonormal μ–rotations per plane rotation, increases the accuracy of the approximate rotation. Therefore, the conditions for quadratic convergence [13, 15] are met as the diagonalization advances. Here, the Jacobi method is executed with an adaptive number r of orthonormal μ–rotations per plane rotation (dashed line), where $r = \lfloor |k_{mean}|/10 \rfloor$ and k_{mean} is the mean value of the angle indices used in the previous sweep ($r = 1$ in the first sweep).

In Figure 5 the required number of shift–add operations per sweep is shown. The Jacobi method using the original (exact) CORDIC requires a constant number of shift–add operations per sweep, since each rotation requires a constant number of shift–add operations (for a 32 bit CORDIC 80 shift–add operations per rotation: 64 for the μ–rotations plus 16 for the scaling of both vector components). The use of approximate rotations, i.e., one orthonormal μ–rotation per plane rotation reduces the number of shift–add operations per sweep significantly. Furthermore, the nature of the Jacobi method guarantees that the less complex μ–rotation methods are used as the matrix becomes more and more diagonally dominant during the course of the algorithm , i.e. ultimately only method I (the least complex method) is required. Therefore, the number of required shift–add operations per sweep reduces during the course of the Jacobi method (solid line). Since the quadratic convergence is lost this must be paid by a greater number of sweeps.

However, this trade–off between number of sweeps and computational cost (number of shift–add operations) works in favour of the approximate rotation scheme. Table 2 shows the total number of sweeps and the total number of shift–add operations required for computing the EVD of the matrix. Figure 6 shows the reduction of the off–diagonal norm vs. the shift–add operations. Obviously, the Jacobi methods using approximate rotations obtain the reduction of the off–diagonal norm with significantly less computational cost, i.e., the overall reduction per shift–add operation is significantly better for the approximate rotations than for the exact rotations. Note, that the exact method actually reduces the off–diagonal norm to $O(10^{-15})$, although the stopping criteria is $10^{-8} \cdot \|\boldsymbol{A}\|_F$. This is due to the fast reduction of the off–diagonal norm when quadratic convergence of the algorithm is reached. Therefore, the Jacobi method with exact rotations actually requires 7 sweeps and 912000 shift–add operations (Figures 4 and 6 are only shown for the actual stopping criteria).

The adaptive scheme works as well as the simple $r = 1$ scheme. When working on a parallel array of processors, the off-chip communication becomes a function of the number of sweeps. In this case it is advantageous to use the adaptive scheme, with less sweeps, as it reduces the off–chip communication.

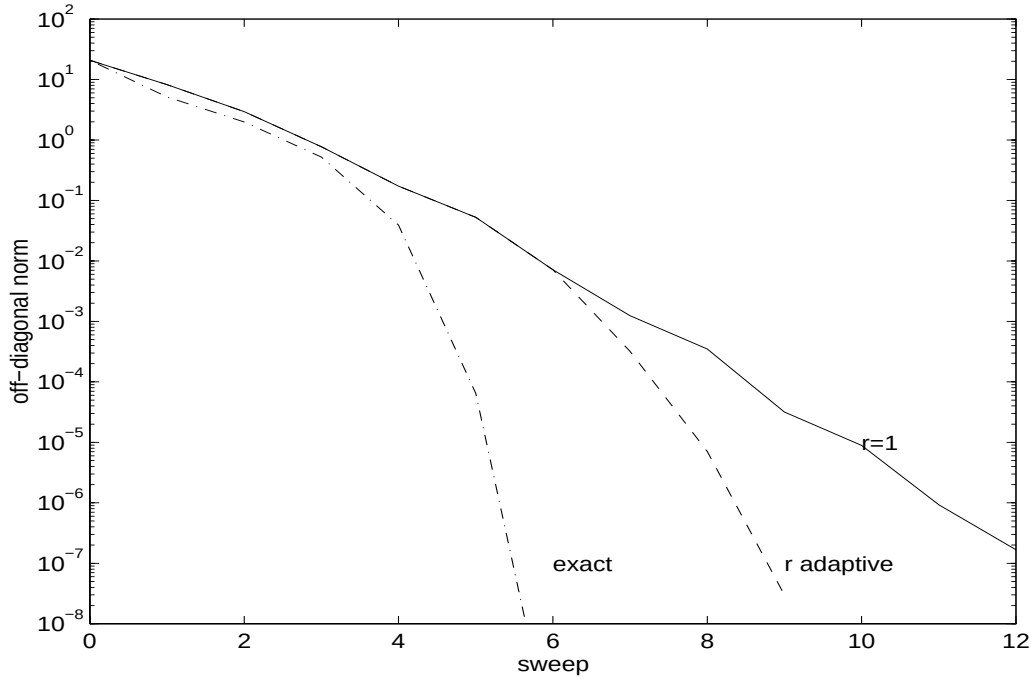

Figure 4: Off–diagonal norm vs. sweeps for the Jacobi method using exact rotations (dash-dotted) and the Jacobi method using approximate rotations (one orthonormal μ–rotation for approximating the exact rotation (solid), r orthonormal μ–rotations for approximating the exact rotation (dashed)).

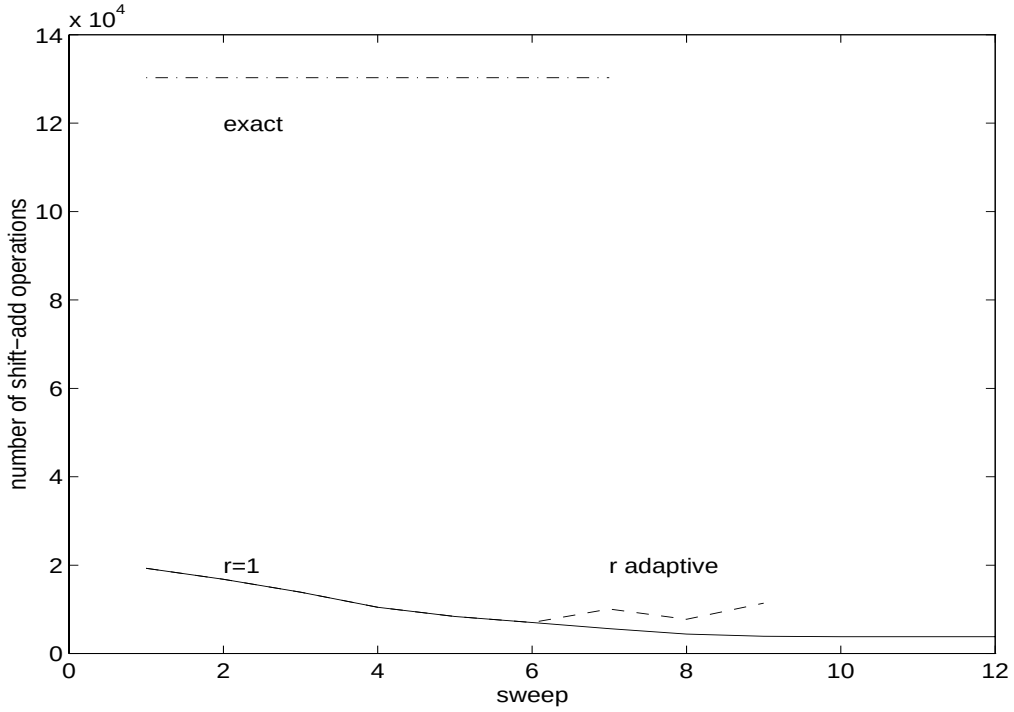

Figure 5: Shift–add operations per sweep for the Jacobi method using exact rotations (dash-dotted) and the Jacobi method using approximate rotations (one orthonormal μ–rotation for approximating the exact rotation (solid), r orthonormal μ–rotations for approximating the exact rotation (dashed)).

Method	exact	$r = 1$	r adaptive
# of sweeps	7	12	9
total # of shift–add	912000	101280	105120

Table 2: Total number of sweeps and total number of shift–add operations.

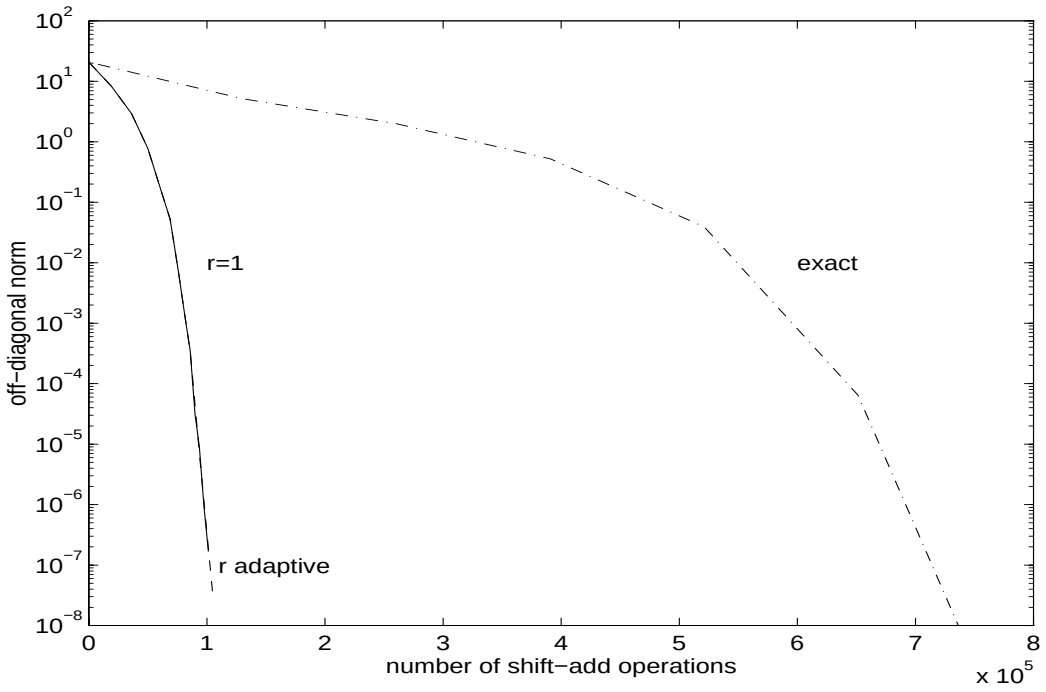

Figure 6: Off–diagonal norm vs. shift–add operations for the Jacobi method using exact rotations (dash-dotted) and the Jacobi method using approximate rotations (one orthonormal μ–rotation for approximating the exact rotation (solid), r orthonormal μ–rotations for approximating the exact rotation (dashed)).

5 REALIZATION OF ORTHONORMAL μ–ROTATIONS

Four basic types of shift–add stages, shown in figure 7a are sufficient to implement any kind of fast rotation.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \boldsymbol{F}_k(\sigma) \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{33}$$

Each stage implements a pair of shift-add operations. In turn, these four types of stages are combined in the unified stage, as shown in figure 7b, which forms the basis of the fast orthonormal μ–rotation architecture.

Method III is used for illustrating the architecture design. Writing out (33) for the orthonormal μ–rotation of method III yields:

$$\begin{aligned} x' &= x \Leftrightarrow \sigma(y \cdot 2^k) \Leftrightarrow (x \cdot 2^{2k-1}) + \sigma(y \cdot 2^{3k-3}) \\ y' &= y + \sigma(x \cdot 2^k) \Leftrightarrow (y \cdot 2^{2k-1}) \Leftrightarrow \sigma(x \cdot 2^{3k-3}) \end{aligned} \tag{34}$$

The rotation of (34) is realized as a cascade of simple stages, as shown in figure 8c. The sequence of shifted values of x, being $\{x \cdot 2^k, x \cdot 2^{2k-1}, x \cdot 2^{3k-3}\}$, are computed through consecutive shifts over $k, k \Leftrightarrow 1$ and $k \Leftrightarrow 2$ positions. The same is done for the shifted values of y. Intermediate shifted results are stored in auxiliary variables x_a, y_a.

Similar sequences exist for the other unscaled rotation methods (I, II) as is shown in figures 8a and 8b, respectively. The scaled rotation method IV consists of a double rotation, implemented as two rotation stages, followed by a variable length scaling sequence. This is shown in figure 8d, for a scaling sequence of length 3.

5.1 Floating-point realization

The realization of an orthonormal μ–rotation of floating-point x, y data over the angle α_k is performed in three stages.

1. floating point preprocessing; alignment of mantissas
2. fixed-point execution of the rotation
3. floating-point postprocessing; renormalisation

We assume a fixed-point datapath to realize the rotation. The x and y summation path *each* has their own fixed-point exponent $e_x^{(dp)}$ resp. $e_y^{(dp)}$.

The first stage is the floating-point pre-processing. From the exponents e_x, e_y and the angle index (or angle *exponent*) k, we compute the exponents $e_x^{(dp)}, e_y^{(dp)}$ which are used inside the fixed point datapath.

$$\begin{aligned} e_x^{(dp)} &= \max(e_x, e_y + k) \\ e_y^{(dp)} &= \max(e_y, e_x + k) \end{aligned} \tag{35}$$

These datapath exponents are chosen such that the final and intermediate results do not overflow the fixed-point datapath, while still maintaining full accuracy. Only one extra MSB bit must be added as a precaution. The mantissas are aligned accordingly before they enter the fixed point datapath.

$$\begin{aligned} m_x^{(dp)} &= m_x \cdot 2^{e_x - e_x^{(dp)}} \\ m_y^{(dp)} &= m_y \cdot 2^{e_y - e_y^{(dp)}} \end{aligned} \tag{36}$$

The second stage is the fixed-point execution of the rotation. Computing in fixed-point implies that the exponents of the intermediate results remain the same during computation. Operations are performed on the mantissas only, greatly speeding up operations since expensive floating-point additions are eliminated. Writing out the rotation in terms of mantissas only, we arrive at the following equation for fixed-point rotation:

$$\begin{aligned} m'_x &= \hat{c} \cdot m_x^{(dp)} \Leftrightarrow \sigma \hat{s} \cdot 2^{\Delta} \cdot m_y^{(dp)} \\ m'_y &= \hat{c} \cdot m_y^{(dp)} + \sigma \hat{s} \cdot 2^{-\Delta} \cdot m_x^{(dp)} \end{aligned} \tag{37}$$

where $\Delta = e_y^{(dp)} \Leftrightarrow e_x^{(dp)}$ is the difference in exponents between the x and y datapath. The above formula is the basis for the same cascade realizations as in figure 8. The only difference in the execution of the cascaded stages is that, for the "Rotation" stages only, the k_x and k_y shifts are offset by Δ resp. $\Leftrightarrow\Delta$. This is necessary to align data transferred from the x to the y datapath resp. from the y to the x datapath. From equations (35) we can prove that Δ remains bounded by $k \leq \Delta \leq \Leftrightarrow k$. This guarantees that only negative shifts (shift right operations) occur during the computation.

The third and last step is the floating-point post-processing. The results of the fixed-point computation is paired with the corresponding exponents as $(m'_x, e_x^{(dp)})$ and $(m'_y, e_y^{(dp)})$ and re-normalised to proper floating-point representation.

5.2 Architecture considerations

Due to the variable length of the different orthonormal μ-rotations, a sequential architecture for the computational core of the processor is highly favourable. The communication between processors in a parallel system, however, must be capable to handle the variations in computation time.

As for the computational core itself, the floating-point sequential CORDIC architecture presented in [19] only needs slight modifications to accommodate the functionality of the unified stage of figure 7b, making it suitable to perform fast, orthonormal μ–rotations, as well as the normal CORDIC operations.

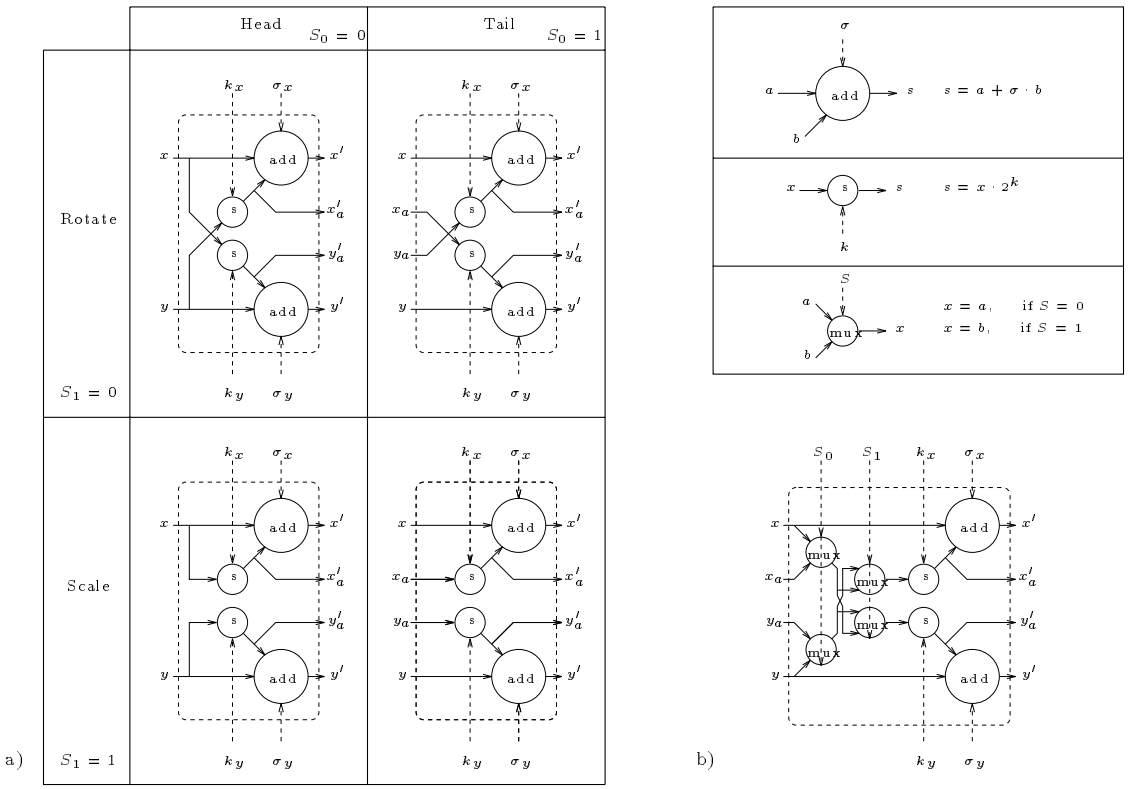

Figure 7: The four basic shift–add stages (a) and their unification (b).

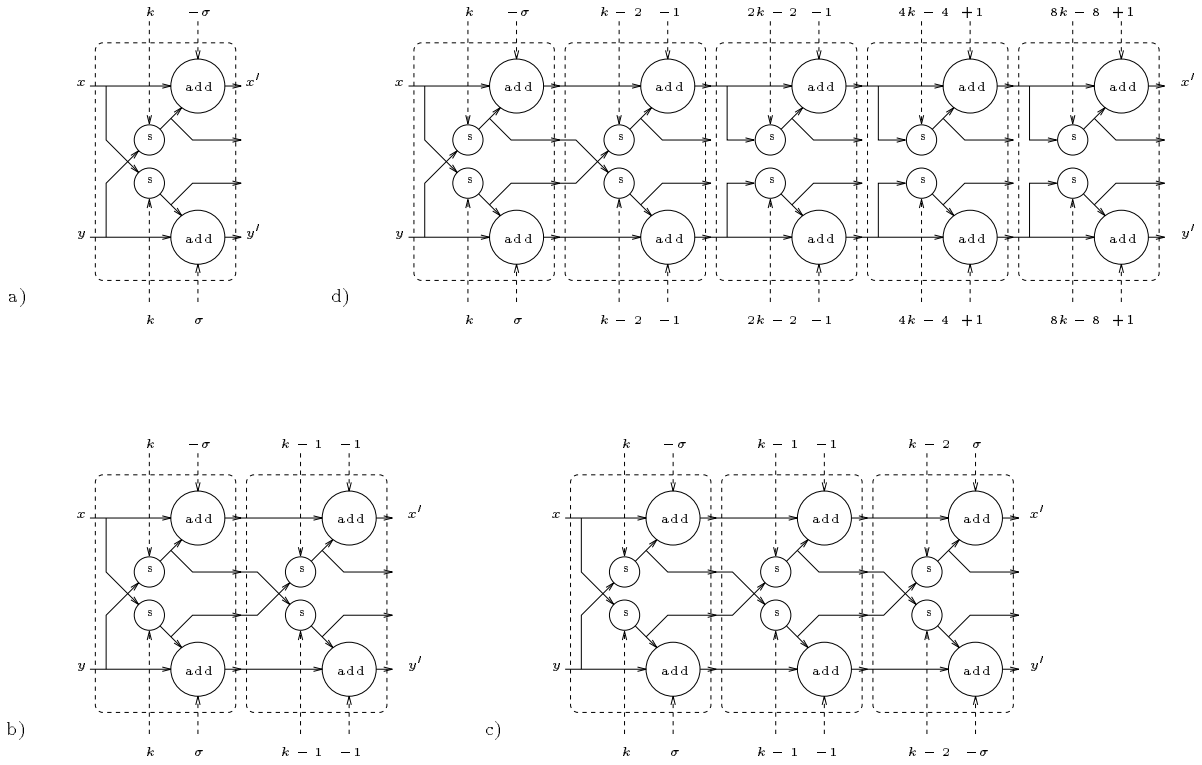

Figure 8: Cascade realization of the orthonormal μ–rotations (method I (a), method II (b), method III (c), method IV with $m = 3$ (d)).

6 CONCLUSIONS

In this paper a Jacobi–type algorithm for computing the EVD of a symmetric matrix was presented. It uses different types of orthonormal μ–rotations as approximate rotations. These orthonormal μ–rotations are distinguished by decreasing complexity for decreasing angles of rotation. The evaluation of the angle index k, which determines the approximate rotation angle and the used type of orthonormal μ–rotation, can also be executed by at most three (unscaled) μ–rotations. Therefore, the entire Jacobi–type method can be performed by the execution of μ–rotations (pairs of shift–add operations, respectively), i.e., it can completely be executed on a floating point CORDIC-like architecture. Thus, it is highly suitable for a VLSI implementation.

Furthermore, the nature of the Jacobi method (i.e. increasing diagonalization $\hat{=}$ decreasing rotation angles) supports the use of different types of orthonormal μ–rotations as well as an adaptation of the accuracy of the approximate rotation, since in both cases the simpler types of μ–rotations are used as the required rotation angle (angle index k) decreases.

Finally, note that the presented methods also apply to the SVD of a rectangular matrix if the SVD problem is mapped to a symmetric EVD problem with increased dimension [18, 20]. The methods can also be applied to the SVD without this mapping procedure [21] by applying the approximate rotation scheme to the CORDIC based SVD methods presented in [22, 23].

ACKNOWLEDGEMENTS

The work of the first author was performed while at Delft University of Technology. Thanks are to the Network Theory Group (Prof. P. Dewilde, Prof. E. Deprettere) of the Delft University of Technology for the hospitality and the great time in Delft.

References

[1] R.O. Schmidt. Multiple emitter location and signal parameter estimation. *IEEE Trans. on Antennas and Propagation*, AP-34:276–280, 1986.

[2] A. Paulraj, R. Roy, and T. Kailath. A subspace approach to signal parameter estimation. *Proceedings of the IEEE*, 74:1044–1045, 1986.

[3] M. Verhaegen and P. Dewilde. Subspace model identification. part I: The output–error state–space model identification class of algorithms. *Int. J. Control*, 56:1187–1210, 1992.

[4] R.P. Brent and F.T. Luk. The solution of singular value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.*, 6:69–84, 1985.

[5] J.-M. Delosme. Bit-level systolic algorithm for the symmetric eigenvalue problem. In *Proc. Int. Conf. on Application Specific Array Processors*, pages 771–781, Princeton (USA), 1990.

[6] J.J. Modi and J.D. Pryce. Efficient implementation of Jacobi's diagonalization method on the DAP. *Numer. Math.*, 46:443–454, 1985.

[7] J. Demmel and K. Veselic. Jacobi's method is more accurate than QR. *SIAM J. Matrix Anal. Appl.*, 13:1204–1245, 1992.

[8] J.P. Charlier, M. Vanbegin, and P. van Dooren. On efficient implementations of Kogbetliantz's algorithm for computing the singular value decomposition. *Numer. Math.*, 52:279–300, 1988.

[9] W.M. Gentleman. Least squares computations by Givens rotations without square roots. *J. Inst. Maths Applics*, 12:329–336, 1973.

[10] W. Rath. Fast Givens rotations for orthogonal similarity transformations. *Numer. Math.*, 40:47–56, 1982.

[11] J. Götze and U. Schwiegelshohn. A square root and division free Givens rotation for solving least squares problems on systolic arrays. *SIAM J. Sci Stat. Comput.*, 12:800–807, 1991.

[12] M.D. Ercegovac and T. Lang. Redundant and on–line CORDIC: Application to matrix triangularization and SVD. *IEEE Trans. on Computers*, 39:725–740, 1990.

[13] J. Götze. On the parallel implementation of Jacobi and Kogbetliantz algorithms. *SIAM J. Sci. Comput.*, 15:1331–1348, 1994.

[14] J. Götze. Parallel methods for iterative matrix computations. In *Proc. IEEE Int. Symp. on Circuits and Systems*, pages 233–236, Singapore, 1991.

[15] J. Götze, S. Paul, and M. Sauer. An efficient Jacobi–like algorithm for parallel eigenvalue computation. *IEEE Trans. on Computers*, 42:1058–1065, 1993.

[16] J. Götze. Monitoring the stage of diagonalization in Jacobi–type methods. In *Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing*, pages III 441–444, Adelaide (Australia), 1994.

[17] J.-M. Delosme. CORDIC algorithms: theory and extensions. In *SPIE Advanced Algorithms and Architectures for Signal processing IV*, volume 1152, pages 131–145, San Diego (USA), 1989.

[18] G.H. Golub and C.F. van Loan. *Matrix Computations*. The John Hopkins University Press, second edition, 1989.

[19] G.J. Hekstra and E.F. Deprettere. Floating point CORDIC. In *11th Symp. on Computer Arithmetic*, Winsor (Canada), 1993.

[20] J. Götze. CORDIC–based approximate rotations for SVD and QRD. In *To appear Proc. European Signal Processing Conference*, Edinburgh (Scotland), 1994.

[21] J. Götze, P. Rieder, and J.A. Nossek. Parallel SVD–updating using approximate rotations. In *SPIE Advanced Signal Processing: Algorithms, Architectures and Implementations VI*, San Diego (USA), 1995.

[22] B. Yang and J.F. Böhme. Reducing the computations of the singular value decomposition array given by Brent and Luk. *SIAM J. Matrix Anal. Appl.*, 12:713–725, 1991.

[23] J.R. Cavallaro and F.T. Luk. CORDIC arithmetic for an SVD processor. *J. Parallel & Distributed Computing*, 5:271–290, 1988.

List of Figures

List of Tables