A Parallel Square-Root Algorithm for Modified Extended Kalman Filter

MI LU, Member, IEEE
XIANGZHEN QIAO
Texas A & M University
GUANRONG CHEN, Member, IEEE
University of Houston

A parallel square-root algorithm together with its systolic arrays implementation are proposed for performing the modified extended Kalman filter (MEKF). The proposed parallel square-root algorithm is designed based on the singular value decomposition (SVD) and the Faddeev algorithm, and a very large scale integration (VLSI) systolic arrays architecture is developed for its implementation. Comparing with other square root Kalman filtering algorithms existing in the literature, the proposed method is more numerically stable. Moreover, the new VLSI architecture has very nice parallel and pipelining characteristics in applying to the MEKF and achieves higher efficiency. For n-dimensional state vector estimations, the proposed architecture consists of \(O(2n^3)\) processing elements and uses \(O((t + 17)n)\) time-steps for a complete iteration at each instant, in contrast to the complexity of \(O((t + 6)n^3)\) time-steps for a sequential implementation, where \(t \approx \log n\).

1. INTRODUCTION

The Kalman filter [11] is an optimal linear estimation algorithm introduced by R. E. Kalman in 1960. Since then, it has been widely applied in academic, industrial, military, and aerospace engineering areas such as systems control, navigation and guidance, signal processing, and communication. The Kalman filter is considered to be an optimal estimator because it provides linear, unbiased, and minimum variance estimate for unknown state vectors of a linear control system. More precisely, under the framework of a state-space description, it provides a real-time algorithm for estimating the state vector of a discrete-time linear system with noise disturbance, using also noisy observation data.

The standard Kalman filter has been designed for linear state-space models. If the model is nonlinear, however, a linearization is necessary. There is a successful technique, known as the extended Kalman filter (EKF), for this purpose. In the EKF, the usual linear Taylor approximations of the nonlinear system and observation equations are used. The EKF is also a real-time algorithm and efficient, but it does not always produce desirable results. Hence, a modified (improved) version of the EKF, called a modified extended Kalman filter (MEKF) is recently introduced [4], in which the center for each up-dated linear Taylor approximation is obtained by using the optimal Kalman filter so that the filtering performance can be very much improved. Roughly speaking, in the EKF the nonlinear model is approximated by a linear model, while in the MEKF the nonlinear model is approximated by two linear models with a better approximation at each time instant during the filtering process. For more details, the reader is referred to the original paper [4].

It has been observed that the standard Kalman filtering algorithm is in many cases numerically unstable. For this reason, several square-root filtering methods [7] were proposed to handle this computational stability problem. On the other hand, the standard Kalman filter is also computationally intensive in the sense that the excessive computational requirements prevent its great potential from many real-time applications. Parallel processing provides a solution to this problem. For example, VLSI (very large scale integration) technology enables us to develop special-purpose processors in combining with some topological structures to obtain supercomputing, with faster throughput rates than traditional sequential computing [12]. In the past few years, some efforts have been made to combine the power of VLSI and new techniques in signal processing. In this endeavor, some recent research have been devoted to finding new algorithms and then mapping them to parallel architectures. As a matter of fact, the concept of the so-called “algorithm engineering” has recently emerged.
This interdisciplinary research objective is first to derive numerical algorithms which are suitable for parallel computations and then to implement them by fast VLSI architectures.

In the investigations of the Kalman filtering algorithm engineering, one may trace back to, for example, Andrews [1] for a simple parallel algorithm of the Kalman filter, where a parallel structure using elementary processors for implementing a square-root algorithm was suggested. Later, Jover and Kailath [10] described a systolic arrays architecture for square-root covariance Kalman filters, using a square-root free QR decomposition technique (fast Givens factorization). Different approaches were then published [7], in which many of them were based on square-root covariance or information Kalman filter processes.

In such architectures, a triangular square-root is propagated to preserve the symmetry of the covariance (or information) matrices for the purpose of numerical stability. In this approach, the Jacobian or Givens rotations for the square-root factorization are frequently used. For example, Phillips and Frabzio [16] applied the systolic implementation of the Runge-Kutta algorithm in their real-time calculation of the EKF process. Babetti et al. [2] investigated an efficient implementation of the EKF on a parallel computer designed for real-time applications. In all these methods, the (positive) square-root of a matrix is calculated by the orthogonal triangularization or the Cholesky decomposition. For an $m \times n$ matrix $A$ (with $n \leq m$), when $\text{rank}(A) = r < n$, this triangularization is not numerically stable. Thus, in many cases, these methods cannot give expected results. To handle this problem, the singular value decomposition (SVD) is desirable.

We investigate the MEKF instead of the EKF. We propose a square-root algorithm based on the SVD to implement the MEKF, in which the Kalman filtering algorithm is adjusted to be one of the Faddeev type algorithm [15]. As is known, the Faddeev algorithm provides a wide range of matrix computational capabilities and can easily map the computational scheme onto a concurrent systolic array architecture. Using this algorithm, a VLSI parallel processing computing system for the MEKF implementation is proposed here. The time needed for a complete iteration of the sequential execution of this method is $O((s + 6)n^2)$, where $n$ is the dimension of the state vector and $s \approx \log n$. However, the parallel architecture that we propose, using $O(2n^3)$ processing elements, only takes $O((s + 17)n)$ time-steps to complete an iteration. Here and throughout the paper, one time-step represents one floating point multiplication and one addition.

The rest of the paper is organized as follows. In Section II, the MEKF algorithm is described. Section III is devoted to a square-root algorithm based on the SVD and the Faddeev algorithm, followed by the proposed parallel implementation of the MEKF in Section IV. Finally in Section V, conclusions are drawn.

II. MODIFIED EXTENDED KALMAN FILTER

The MEKF is introduced in [4] to improve the performance of the EKF. The nonlinear model under consideration and the corresponding MEKF can be described as follows. Let $x_k$ and $y_k$ be an $n$- and $m$-dimensional vector, respectively, and let the state vector of the system be the $(n + m)$-dimensional vector $[\xi_k^T \eta_k^T]^T$. The following equations give the state-variable description for the nonlinear system under investigation:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} F_k(y_k)x_k \\ H_k(x_k,y_k) \end{bmatrix} + \begin{bmatrix} \Gamma_k^1(x_k,y_k) & 0 \\ \Gamma_k^2(x_k,y_k) & \Gamma_k^3(x_k,y_k) \end{bmatrix} \begin{bmatrix} \xi_k^T \\ \eta_k \end{bmatrix},$$

where $\{\xi_k^T \}$ and $\{\eta_k \}$ are uncorrelated (zero-mean Gaussian) white noise sequences with covariance matrices

$$Q_k = \text{var} \left( \begin{bmatrix} \xi_k^T \\ \xi_k \end{bmatrix} \right)$$

and $R_k = \text{var}(\eta_k)$, respectively. $F_k$, $H_k$, $C_k$, $\Gamma_k^1$, $\Gamma_k^2$, $\Gamma_k^3$ are nonlinear matrix-valued functions, $F_k$ and $C_k$ are assumed to be differentiable, $\xi_k \in \mathbb{R}^p$, $i = 1, 2,$ and $p = p_1 + p_2$ not exceeding $N$ ($N = n + m$).

The basic idea of the EKF is to linearize a nonlinear model in real time by using each previous suboptimal estimate as the center for the updated linear Taylor approximation. However, the MEKF uses an improved real-time linearization procedure in the sense that the center for each updated linear Taylor approximation is derived from an optimal Kalman filtering algorithm [4]. Thus, the MEKF process for the above nonlinear model consists of two subsystems, which corresponds, respectively, to the following two state-space descriptions:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_k} F_k(y_k)x_k \\ \frac{\partial}{\partial y_k} F_k(y_k)x_k \end{bmatrix} \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} \Gamma_k^1(x_k,y_k) & 0 \\ \Gamma_k^2(x_k,y_k) & \Gamma_k^3(x_k,y_k) \end{bmatrix} \begin{bmatrix} \xi_k^T \\ \eta_k \end{bmatrix},$$

where $\{\xi_k^T \}$ and $\{\eta_k \}$ are uncorrelated (zero-mean Gaussian) white noise sequences with covariance matrices

$$Q_k = \text{var} \left( \begin{bmatrix} \xi_k^T \\ \xi_k \end{bmatrix} \right)$$

and $R_k = \text{var}(\eta_k)$, respectively. $F_k$, $H_k$, $C_k$, $\Gamma_k^1$, $\Gamma_k^2$, $\Gamma_k^3$ are nonlinear matrix-valued functions, $F_k$ and $C_k$ are assumed to be differentiable, $\xi_k \in \mathbb{R}^p$, $i = 1, 2,$ and $p = p_1 + p_2$ not exceeding $N$ ($N = n + m$).
and

\[
\begin{align*}
    x_{k+1} &= F_k(\hat{x}_k)x_k + \Gamma_k^1(\hat{x}_k, \hat{y}_k)\xi_k^1 \\
    v_k &= C_k(\hat{x}_k, \hat{y}_k)x_k + \eta_k
\end{align*}
\]  

(3)

where \( [\hat{x}_k^T, \hat{y}_k^T] \) and \( \hat{x}_k \) are obtained parallelly as shown in Fig. 1 (see [4] for more details) and the constant vector

\[
u_k = \left[ \frac{F_k(\hat{y}_k)\hat{x}_k}{H_k(\hat{x}_k, \hat{y}_k)} \right] \left[ \frac{\partial}{\partial x_k} \left[ \frac{F_k(\hat{y}_k)\hat{x}_k}{H_k(\hat{x}_k, \hat{y}_k)} \right] \right] \hat{y}_k^T \]

can be treated as a deterministic control input. Here, the Jacobian matrix is defined in a standard way:

\[
\begin{bmatrix}
    \frac{\partial}{\partial x_k} \left[ \frac{F_k(\hat{y}_k)\hat{x}_k}{H_k(\hat{x}_k, \hat{y}_k)} \right] \\
    \frac{\partial}{\partial y_k} \left[ \frac{F_k(\hat{y}_k)\hat{x}_k}{H_k(\hat{x}_k, \hat{y}_k)} \right]
\end{bmatrix}
\]

Thus, in the MEKF the nonlinear model (1) provides two standard linear systems (2) and (3) with different dimensions, at each time instant during the filtering process. The two models (2) and (3) can be solved in parallel starting with the same initial estimate as shown in Fig. 1. Without taking computational issues into account, these two linear systems may be solved to yield optimal estimates by simply using the standard Kalman filtering algorithm [5].

As mentioned above in Section I, in order to obtain a numerically stable and computationally efficient parallel architecture for the MEKF, we propose to use the square-root Kalman filtering algorithm based on the SVD and the Faddeev scheme. To describe this new parallel implementation, we first discuss the SVD-based square-root Kalman filtering algorithm in the following section.

III. SVD-BASED SQUARE-ROOT KALMAN FILTERING ALGORITHM

The Kalman filter provides a real-time algorithm for estimating the \( n \)-dimensional state vector \( x_k \) of a discrete-time linear system

\[
    x_{k+1} = A_k x_k + \Gamma_k \xi_k
\]

given a \( q \)-dimensional observation vector \( v_k \)

\[
    v_k = C_k x_k + \eta_k
\]

(4)

(5)

where \( \xi_k \) and \( \eta_k \) are two sequences of uncorrelated (zero-mean Gaussian) white noise with \( \text{var}(\xi_k) = Q_k \), \( p \times p \) nonnegative definite matrix, \( \text{var}(\eta_k) = R_k \), \( a \times q \) positive definite matrix, and \( E(\xi_k \eta_k^T) = 0 \) for all \( k \) and \( l \). In the above, \( A_k, \Gamma_k, \) and \( C_k \) are known matrices of dimensions \( n \times n, n \times p, \) and \( q \times n \), respectively. The initial state \( x_0 \) is also assumed to be uncorrelated with \( \xi_k \) and \( \eta_k \) in the sense that \( E(x_0 \xi_k^T) = 0 \) and \( E(x_0 \eta_k^T) = 0 \) for all \( k \).

The standard Kalman filter estimates the state vectors of the system from a sequence of measurements. The estimates \( \hat{x}_k \) are updated by using the following formulas:

\[
P_{0,0} = \text{var}(x_0)
\]

\[
P_{k,k-1} = A_k^{-1} P_{k-1,k-1} A_k^{-1} + \Gamma_k^{-1} Q_k^{-1} \Gamma_k^{-1}
\]

(6)

\[
    G_k = P_{k,k-1} C_k^T (C_k P_{k,k-1} C_k^T + R_k)^{-1}
\]

(7)

\[
P_{k,k} = (I - G_k C_k) P_{k,k-1}
\]

(8)

\[
    \hat{x}_{0,0} = E(x_0)
\]

(9)

\[
    \hat{x}_{k,k-1} = A_k^{-1} \hat{x}_{k-1,k-1}
\]

(10)

\[
    \hat{x}_{k,k} = \hat{x}_{k,k-1} + G_k (v_k - C_k \hat{x}_{k,k-1})
\]

(11)

Thus, the Kalman gain matrices \( G_k \) and \( P_{k,k} \) are the updated estimates and error covariances.

In general, this filtering algorithm is numerically unstable. For this reason, several different square-root algorithms have been proposed. For example, the so-called information square-root algorithm is a popular one [7]. In the information square-root algorithm, it is assumed that the matrices of which square-roots are taken are all positive definite. If this condition is satisfied, then the Cholesky factorization can be applied. However, as is known, many of the matrices which need to be taken square-roots in the Kalman filtering process are only nonnegative definite but never positive definite, so that this approach is not applicable. On the other hand, there is a so-called covariance square-root algorithm, in which (6) is expressed as

\[
[A_{k-1} P_{k-1,k-1}^{1/2} | \Gamma_{k-1} Q_{k-1}^{1/2}] H = [P_{k,k-1}^{1/2} | 0]
\]

(12)

where \( H \) is an orthogonal matrix. This factorization depends on a general orthogonal transformation or the Cholesky decomposition. In this approach, the computations are not necessarily stable and efficient, particularly when the matrix to be factorized is not square.
A more stable and efficient square-root algorithm seems to be the one based on the SVD which we study below.

By a “square-root” of a matrix we mean the following: For any \(n \times (n + p)\) matrix \(A\), there is always an \(n \times n\) matrix \(\tilde{A}\) such that \(A\tilde{A} = AA'\). \(\tilde{A}\) is called a square-root of the matrix \(AA'\).

Here, we notice that a square-root of the matrix \(AA'\) is not necessarily lower triangular nor positive definite. According to the above definition, the square-root Kalman filtering algorithm discussed in [5] can be written as follows.

1) Compute \(J_{0,0} = (\text{var}(x_0))^{1/2}\) and \(\hat{x}_{0,0} = E(x_0)\).
2) For \(k = 1, 2, \ldots\),
   a) compute \(J_{k,k-1}\), the square-root of the matrix \(J_{k-1,k-1} = [A_{k-1,k-1} G_{k-1,k-1}^2] [A_{k-1,k-1} G_{k-1,k-1}]'\), then compute \(H_k = (C_k J_{k-1,k-1} C_k') + R_k\) and \(\tilde{x}_{k,k-1} = J_{k,k-1} [I - J_{k-1,k-1} C_k (H_k + R_k)^{-1} C_k J_{k-1,k-1}]\).
   b) Using the information from a), compute \(G_k = J_{k,k-1} J_{k-1,k-1} C_k (H_k + R_k)^{-1} H_k^{-1} \) and \(\tilde{x}_{k,k} = A_{k-1,k-1} \tilde{x}_{k,k-1} + G_k (v_k - C_k \tilde{x}_{k,k-1} \tilde{x}_{k,k-1})\).

In the above, the superscript \(c\) indicates the Cholesky decomposition of a positive-definite matrix, which is lower triangular and hence very easy to invert.

Taking into account the numerical stability, we give the following remarks, discussions, and suggestions on the implementation of the above algorithm.

1) The computation of \(J_{k,k-1}\): Let \(B = [A_{k-1,k-1} G_{k-1,k-1}^2]\), which is an \(n \times (n + p)\) real matrix. When \(r = \text{rank}(B) = n \leq (n + p)\), it is always possible to factorize \(B\) by using some orthogonal transformation: \(B = QR\), where \(Q\) is orthogonal and \(R\) is upper triangular. In this case, many existing results such as those mentioned in Section I can be used. In the case where the rank is not full, that is, when \(r < n\), the \(QR\) factorization does not necessarily produce an orthogonal basis for the linear span of the matrix \(B\). In this case, there is a simple method for solving the problem, which is the \(QR\) factorization with column pivoting. Unfortunately, as a method for handling near rank deficiency, the pivoting \(QR\) method is not completely reliable either [9]. The most reliable technique to take care of the rank deficiency seems to be the SVD. More precisely, it is preferable to develop a method using the SVD to calculate the matrix \(J_{k,k-1}\), a square root of \(BB'\). It may seem to be obvious that \(J_{k,k-1}\) can be obtained by first computing \(C = BB'\) and then computing a square root of \(C\). However, as is known, forming the matrix \(BB'\) may lead to a loss of information which may then cause numerical instability [9, p. 289]. Hence, we propose an SVD approach as described in the following.

The SVD of an \(n \times m\) matrix \(B\) is defined as follows:

\[
B = U \Sigma V'
\]

where \(U = [u_1, u_2, \ldots, u_m] \in \mathbb{R}^{n \times m}\) and \(V = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{n \times n}\) are orthogonal matrices and \(\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}\) is a real nonnegative “diagonal” matrix. Since \(B' = B \Sigma V'\), without loss of generality we may assume that \(n \leq m\). The square root of \(BB'\) can be computed from the SVD of \(B'\):

\[
BB' = V \Sigma V'
\]

Let

\[
B_r = V \begin{bmatrix}
\sigma_1 & 0 & \cdots & 0 \\
0 & \sigma_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_n
\end{bmatrix} V'
\]

Then \(B_r\) is an \(n \times n\) square matrix, which satisfies

\[
B_r B_r' = V \begin{bmatrix}
\sigma_1 & 0 & \cdots & 0 \\
0 & \sigma_2 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & \sigma_n
\end{bmatrix} V' = BB'.
\]

Hence, the matrix \(B_r\) is a square root of the matrix \(BB'\), and it can be calculated from the SVD of either the matrix \(B\) or \(B'\).

2) The SVD of the \(m \times n\) Matrix \(B'\) (\(n \leq m\)):
There are some important relationships between the SVDs of an \(n \times m\) \((n \leq m)\) matrix \(B\) and the \(QR\) orthogonal decomposition of the symmetric matrices \(BB'\) and \(B'\). If

\[
U' B' V' = \Sigma' = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{m \times n}
\]

is the SVD of \(B'\) in \(\mathbb{R}^{n \times m}\), where \(U \in \mathbb{R}^{m \times m}\) and \(V \in \mathbb{R}^{n \times n}\) are orthogonal matrices, then we have

\[
B_r V' = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}
\]

and

\[
U'(BB')U = \text{diag}(\sigma_1^2, \ldots, \sigma_n^2, 0, \ldots, 0) \in \mathbb{R}^{m \times m}.
\]

A variant of the \(QR\) algorithm can be used to calculate the SVD of a given matrix. At the first glance, this seems to be straightforward. The SVD of \(B'\) can be obtained by the computation of the eigenvalues of \(BB'\). However, as mentioned above, forming the
matrix $BB'$ may lead to a loss of information. A practical method for computing the SVD of $B'$ is to find $U$ and $V$ simultaneously by implicitly applying the symmetric QR algorithm to $B'$. For calculating the SVD of a square $n \times n$ matrix $B'$, we use the extended Jacobian eigenvalue algorithm. This can be done by diagonalizing $B'$ via a sequence of two-by-two SVDs. This algorithm computes an SVD in $O(sm/n^2)$ time-steps, where $s$ is a slowly growing function of $n$ which is conjectured to be $O(\log n)$. Usually, $m$ is considered as a constant. When $m > n$, we can use the following strategy to handle the rectangular SVD problem. First, we compute the $QR$ factorization of $B'$:

$$B' = Q \begin{bmatrix} R \\ 0 \end{bmatrix}$$

where $R \in \mathbb{R}^{n \times n}$ is upper triangular. Then we compute the square SVD of $R$:

$$W' RV = \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}.$$  \hspace{1cm} (13)

By defining

$$U = Q \begin{bmatrix} W & 0 \\ 0 & I \end{bmatrix}$$

we obtain

$$U' B' V = \Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^{m \times n}.$$  \hspace{1cm} (15)

This process computes the SVD in $O(sm^2 + mn^2)$ time-steps.

3) The Faddeev Algorithm and the Computation of $J_{k,k}$: The Faddeev algorithm [15] provides a wide range of matrix computation capability, which can be easily mapped to systolic array architectures. The Faddeev algorithm can also be easily systematized for matrix calculations in such a way that it is highly desirable from an architecture point of view.

Observe that the computation of the matrix $J_{k,k}$ is based on some basic linear algebraic operations, such as matrix-matrix multiplications and additions, matrix-vector multiplications, and matrix inversions. Therefore, in order to maximize the capability of the hardware implementation of systolic arrays, it is very natural to arrange the computation of $J_{k,k}$ into a form suitable for using the Faddeev algorithm. The Faddeev algorithm can be illustrated as follows. Assume that $A$, $B$, $C$, and $D$ are rectangular matrices with dimensions $m \times n$, $n \times p$, $m_2 \times n$ and $m_2 \times p$, respectively, given in the form

$$\begin{bmatrix} A & B \\ -C & D \end{bmatrix}.$$

It is clear that we can perform row operations from the matrices $A$ and $B$ to $-C$ and $D$ and obtain a new matrix as follows:

$$\begin{bmatrix} A & B \\ -C + W_A & D + W_B \end{bmatrix},$$  \hspace{1cm} (16)

where $W$ is a matrix representing the row operations.

Fortunately, in our algorithm, matrix $A$ is always nonsingular and, in fact, upper triangular. This computation needs $m_2n(n/2 + p)$ time-steps. Different matrix operations are possible by selecting appropriate entries in the four quadrants of the matrix. Thus, the Faddeev algorithm is programmable by positioning the data appropriately before the calculation begins. Comparing with other standard matrix computation methods, the Faddeev algorithm has the advantage of considerably saving time and space in the sense of computational complexity. In our proposed implementation, we use the Faddeev algorithm to calculate the following matrix computations: 1) matrix multiplications, 2) matrix inversions and multiplications, 3) matrix multiplications and additions, and 4) linear combinations of matrix operations. Different choices of the entries of this block matrix are listed in Table I. Thus, the computation of $J_{k,k}$ can be divided into a series of Faddeev operations ($A^{-1}B$, $CB$, $D + CB$, $D + CA^{-1}B$ and $CA^{-1}$) and some other basic matrix operations (the Cholesky factorization and matrix-vector multiplications). Based on the above analysis, a complete SVD-based computational scheme for solving the square-root Kalman filtering problem is described in the following algorithm.

Algorithm
1) Compute $J_{k,k-1}$.

   a) Calculate $X_1 = A_{k-2} J_{k-1,k-1} J_{k-1,k-1}^{-1}$ using the Faddeev algorithm, where $C = A_{k-1}$, $B = J_{k-1,k-1}$, $A = I$, $D = 0$.

   b) Calculate $X_2 = Q_{k-1}^{1/2}$ using the Faddeev algorithm, where $C = I_{k-1}$, $B = Q_{k-1}^{-1/2}$, $A = I$, $D = 0$.

   c) Let $B = [X_1, X_2]$, perform the SVD for $B'$:

   $$B' = U \Sigma V',$$

   and find a square root of $BB'$: $J_{k,k-1} = V \Sigma$, where $V$, $U$, $\Sigma$ and $\Sigma$ are defined as in (13)-(15), with $\Sigma = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_n) \in \mathbb{R}^{n \times n}$.

2) Compute $H_k$.  

<table>
<thead>
<tr>
<th>Table I: Faddeev Algorithm for Matrix Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Results of $D$</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>$CB$</td>
</tr>
<tr>
<td>$A^{-1}B$</td>
</tr>
<tr>
<td>$CA^{-1}B + D$</td>
</tr>
<tr>
<td>$CB + D$</td>
</tr>
<tr>
<td>$CA^{-1}$</td>
</tr>
</tbody>
</table>

LU, ET AL.: PARALLEL SQUARE-ROOT ALGORITHM FOR MODIFIED EXTENDED KALMAN FILTER
matrix-matrix multiplications, $MV$ the matrix-vector multiplications, TRAP the computation of the Faddeev algorithm, SVD the computation of an SVD (including the QR factorization), and $LL'$ the Cholesky factorization. It is clear now that this algorithm is very suitable for systolic architecture implementation.

IV. PARALLEL COMPUTATION OF MEKF

In this section, we describe a parallel computation of the MEKF. A new parallel algorithm and parallel architecture are proposed and their performance is analyzed.

A. Parallel Algorithm for MEKF

For implementing the MEKF, different methods are possible. As mentioned above, taking into account the numerical stability, we propose to use the square-root Kalman filter method studied in Section III to implement the MEKF.

The proposed algorithm consists of the following computations: SVDs, matrix-vector multiplications, matrix-matrix multiplications, the Cholesky factorization, and the Faddeev algorithm. All of these computations are so designed that they have highly parallel structure and can be efficiently implemented by systolic arrays. It is known that many matrix operations can be implemented by the Faddeev algorithm. In our design, only one chip is needed to calculate different matrix operations. This strategy will enhance the efficiency of the systolic arrays. From the dependence relations of the algorithm shown in Fig. 2, it can be seen that we only need one SVD chip, one $MV$ chip, one $LL'$ chip, and two TRAP chips.

B. Parallel Architecture for MEKF

We now describe a parallel architecture for the MEKF computations. We first give some preliminaries of parallel structures for matrix operations and then introduce the proposed system architecture.

1) Implementation Considerations: The proposed implementation of the MEKF is based on highly parallel VLSI systolic computing structures, which are very efficient for matrix computations. The following are the matrix operations used to accelerate the square-root Kalman filtering process discussed in the last section: matrix-vector multiplications, the Faddeev algorithm, the Cholesky factorization, and the QR factorization and the SVD.

a) A parallel structure for matrix-vector multiplications: Matrix-vector multiplications can be described by the following recursive relations:

$$y_i = \sum_{k=1}^{n} A_{ik} x_k, \quad (1 \leq i \leq m)$$
where \( A = (A_{i,j}) \) is an \( m \times n \) matrix, \( y \) and \( x \) are \( m \)- and \( n \)-dimensional vectors, respectively. This calculation can be implemented by a linear systolic array (Fig. 3), which needs \( n \) processing cells and takes \( m + n \) time-steps to perform a matrix-vector multiplication.

b) A parallel structure for the Faddeev algorithm:

The Faddeev algorithm can be used to solve many basic matrix operations or combinations of basic matrix operations. Recall the notation introduced in Section III. Suppose that the following compound matrix has been created from the matrices \( A, B, C, \) and \( D: \)
\[
\begin{bmatrix}
A & B \\
-C + WA & D + WB
\end{bmatrix}
\]
and then let \( W \) be chosen so that \( -C + WA = 0 \). Obviously, such a \( W \) is given by \( W = CA^{-1} \), under the assumption that \( A \) is nonsingular. Matrices \( A, B, C, \) and \( D \) are chosen so that many basic matrix operations or combination of basic matrix operations can be executed as indicated in Table I. If matrix \( A \) is first triangularized, then \( A^{-1} \) can be easily calculated to yield \( W = CA^{-1} \). It is clear that this computation consists of two stages. First, matrix \( A \) is transformed to a triangular matrix \( T \). Secondly, \( C \) is annihilated by the Gaussian elimination using the diagonal elements of the matrix \( T \) as pivot elements. In our application, matrix \( A \) is always positive definite and triangular. Thus, only the second stage is needed, in which the Gaussian elimination for eliminating \( C \) is reliable. This process consists of \( m_2 \) steps. The following equations describe the \( k \)th \( (1 \leq k \leq m_2) \) step in this computation:
\[
C_{i,j} = C_{i,j} - C_{i,k}A_{k,j}/A_{k,k},
\]
\[
1 \leq i \leq m_2, \quad k + 1 \leq j \leq n
\]
\[
D_{i,j} = D_{i,j} - C_{i,k}B_{k,j}/A_{k,k},
\]
\[
1 \leq i \leq m_2, \quad 1 \leq j \leq p.
\]

The systolic implementation of the Faddeev algorithm is a trapezoidal array, which is described in Fig. 4. The boundary PEs are used to calculate the coefficients to be eliminated, and the internal PEs to execute the elimination process. Matrices \( A \) and \( B \) are initialized in the array. Matrices \( C \) and \( D \) enter into the array row by row from the top, and then flow down through the array in a skewed form. The computation results then flow out from the bottom of the array. This architecture consists of \( n^2/2 + np \) processing elements and needs \( 2n + m_2 + p \) time-steps to complete the computation.

c) A parallel structure for the Cholesky factorization:

For a positive definite and symmetric matrix \( A = [a_{i,j}] \in \mathbb{R}^{n \times n} \), its Cholesky factorization \( A' = [l_{i,j}] \in \mathbb{R}^{n \times n} \) can be described by the following equations [5]:
\[
l_{i,j} = \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k}^2\right)^{1/2}, \quad i = 1,2,\ldots,n,
\]
\[
l_{i,j} = \left(a_{i,j} - \sum_{k=1}^{i-1} l_{i,k}l_{j,k}\right)/l_{i,j}, \quad j = 1,2,\ldots,i-1, \quad i = 2,3,\ldots,n,
\]
\[
l_{i,j} = 0, \quad j = i + 1,\ldots,n, \quad i = 1,2,\ldots,n.
\]

This computation is similar to the matrix \( LU \) factorization, but a square root is taken during the computation. A triangular systolic array can be used to implement the \( LL' \) factorization. The elements are initialized to be zero in the triangular array. Elements of the matrix, which are to be transferred, enter into the array row by row from the top in a skewed form. The internal cells here have the same function as those in the TRAP. The boundary cells, however, perform more complicated computations as shown in Fig. 5. After the computation, the elements of the lower triangular matrix \( A' (L) \) flow out from the left side of the array. The dataflow is also given in Fig. 5. This architecture consists of \( n^2/2 \) processing elements and needs \( 3n \) time-steps to complete the factorization.
d) A parallel structure for the QR factorization and the SVD: The SVD of an $m \times n$ matrix is given by $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are two orthogonal matrices, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix whose diagonal elements are the singular values of matrix $A$. Different algorithms may be used to solve the SVD problem. The most effective parallel SVD algorithm for a full dense matrix $A$ is probably the two-sided Jacobi algorithm [9]. If matrix $A$ is an $n \times n$ square matrix, then based on an extension of the Jacobi eigenvalue algorithm, we can diagonalize $A$ by a sequence of two-sided rotations. This is discussed in the following.

If $m > n$, then the following approach for handling a rectangular SVD is efficient. The idea is to compute the QR factorization of matrix $A$ first and then to compute a square SVD (see Section III). Usually, this process needs two different algorithmic approaches. However, the systolic array for solving the SVD can also be used for the QR factorization [13]. The algorithm discussed here for triangularizing an $m \times n$ matrix is similar to that described by Gentleman and Kung [8]. The main difference is that instead of performing $2 \times 1$ QR decompositions, we are doing $2 \times 2$ QR decompositions here. The basic transformation for a $2 \times 2$ QR decomposition is a plane rotation

$$J(\theta) = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}.$$

It is chosen such that

$$J(\theta) \begin{bmatrix} w \\ x \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ r \end{bmatrix}.$$

The systolic array in Fig. 6 can be used for the QR factorization. This array consists of a triangular array of $\frac{1}{4}n^2 + O(n)$ processing elements and a triangular array of storage cells. Each cell contains a $2 \times 1$ matrix. Each processor is associated with a $2 \times 2$ matrix contained in the two storage cells at its two upper corners. The QR algorithm needs $m + n - 1$ time-steps. Elements of the matrix $A$ enter into the array row by row from the top. The data inputs as pairs of columns are fed into each processor, with successive pairs of columns being delayed one time unit at each successive processor. The data then flow through the array and each PE performs a rotation. The diagonal processors determine $J(\theta)$ and the others perform rotations. This process is illustrated in Fig. 6. At the end of the process, elements of the triangular matrix $R$ are located in the array. When the eigenvectors are desired to be obtained, matrix $Q'$ needs to be calculated. This can be done by extending the triangular array to a rectangular one. This array needs $\frac{1}{4}mn$ processing elements and $O(m + n)$

time-steps.

The basic computation for an SVD also relates to a $2 \times 2$ matrix. Two plane rotations, $J(\theta)$ and $K(\phi)$, are used to diagonalize a given upper triangular matrix:

$$J(\theta) \begin{bmatrix} w \\ x \end{bmatrix} K(\phi) = \begin{bmatrix} p \\ 0 \\ 0 \\ r \end{bmatrix}.$$
for $i = 2, 4, \ldots$ (i even) do $R = R_{i+1;i} R K_{i+1;i}$.

end.

Usually, iterations stop after $s$ steps, where $s \approx \log n$. This computation can be implemented by a triangular array, which is illustrated in Fig. 7. This array consists of $\frac{1}{2} n^2 + O(n)$ processing elements. Each processing element is associated with a $2 \times 2$ matrix stored in its four neighbor cells. When the eigenvectors are needed, this array is extended to a rectangular one. It consists of $\frac{1}{2} nm$ computing elements. The time needed for an SVD of an $m \times n$ matrix is $O(m + ns)$. It follows from the above analysis that the computations of the QR factorization and the SVD can be implemented by one systolic array [6]. The requirements are $O(m + ns)$ time-steps with $\frac{1}{2} nm$ processing elements.

2) System Architecture for the MEKF Computations: As described in Section II, the MEKF can be performed by using two Kalman filters, which are cooperated in the way shown in Fig. 1. The process for performing the MEKF is illustrated in Fig. 8. Array 1 ($KF_1$) and array 2 ($KF_2$) have the same structure but with different dimensions. $KF_2$ is used for solving (3). This equation describes a linear system. $KF_2$ gives the optimal estimate $\hat{x}_k$ using the input $[\hat{x}_{k-1;1}, \hat{y}_{k-1;1}]^T$ obtained from $KF_1$. $KF_1$ is used for solving (2). It yields the estimates $[\hat{x}_{k;1}, \hat{y}_{k;1}]^T$ using the input $[\hat{x}_{k-1;1}, \hat{y}_{k-1;1}]^T$ obtained from $KF_2$. The dataflow and the structures of array 1 and array 2 are shown in Fig. 9. The two arrays can work in parallel. While array 1 calculates the estimate $[\hat{x}_{k;1}, \hat{y}_{k;1}]^T$ using the input $\hat{x}_{k-1} \hat{y}_{k-1}$ obtained from array 2 at instant $k - 1$, array 2 calculates the estimate $\hat{x}_k$ using the input $[\hat{x}_{k-1;1}, \hat{y}_{k-1;1}]^T$ obtained from array 1 also at instant $k - 1$. After these estimates are calculated, array 1 and array 2 exchange their results. Then the $(k + 1)$th step starts. In this way, array 1 and array 2 work in parallel. In Fig. 9, boxes $MV$, $SVD$, $LL^T$, and $TRAP$ are defined as above in Section III. Since the calculations of $G_k$ and $H_k$ have an overlapping with the calculations of $J_{k,k-1}$ and $H_k$, the calculation of $G_k$ can use the chip $TRAP_1$. Since matrix $R_k^T$ can be calculated while $J_{k,k-1}$ is computed, the two Cholesky factorizations can use the same $LL^T$ chip. Thus, for array 1, this architecture uses $2N^2 + (q^2 + \tilde{q}^2)/2 + N\tilde{q} + Np/2 + N$ processing elements ($\tilde{q} = \max(p, q)$), in which $(N^2 + Np)/2$ processing elements for SVD, $3N^2/2 + \tilde{q}^2/2 + N\tilde{q}$ for the Faddeev algorithm, $q^2/2$ for the $LL^T$ factorization, and $N$ for the matrix-vector multiplications. For array 2, it needs $2n^2 + (q^2 + \tilde{q}^2)/2 + n\tilde{q} + np/2 + n$ processing elements, where $\tilde{q} = \max(p, q)$. The computation for one complete iteration requires $(s + 17)N + 11q + p$ time-steps for array 1 and $(s + 17)n + 11q + p_1$ for array 2. In contrast, the time-step needed for the sequential MEKF is $(s + 6)N^3 + O(N)$. Here, the time-step needed for one complete iteration depends

Fig. 7. Systolic array for SVD.

Fig. 8. Architecture for MEKF.

Fig. 9. Parallel structure for solving square-root Kalman filter.
on array 1. The efficiency depends on the ratio of $n$ and $m$.

V. CONCLUSIONS

A parallel square-root algorithm based on the SVD and the Faddeev scheme has been developed, and a new parallel architecture for implementing the MEKF has also been proposed. Moreover, the performance of the proposed implementation has been analyzed. Comparing with other square-root Kalman filtering algorithms existing in the literature, our method is more numerically stable. The new implementation proposed here is not only suitable for the MEKF but also applicable to the standard (square-root) Kalman filter and EKF, with better numerical stabilities. To develop better pipelining characteristics and to maximize the parallelism, we have used VLSI processor arrays of the systolic type. In the design of the architecture, matrix operations have also been accelerated to speed up the filtering process. For $n$-dimensional state vector estimations, this architecture uses $O(2n^2)$ processing elements and needs $O((s + 17)n)$ time-steps to complete an iteration, in contrast to the complexity of $O((s + 6)n^3)$ time-steps for a sequential implementation, where $s \approx \log n$.

REFERENCES

Parallel processing of the Kalman filter.

Mapping extended Kalman filters onto linear arrays.

Computation of the singular value decomposition using mesh-connected processors.

Modified extended Kalman filtering and a real-time parallel algorithm for system parameter identification.

Kalman Filtering with Real-Time Applications.

SLAPP: a systolic linear algebra parallel processor.
Computer, 23, 7 (Jul. 1987), 45–49.

Systolic Kalman filtering: An overview.
IEE Proceedings-D, 137, 4 (July 1990), 235–244.

Matrix triangularization by systolic arrays.

Matrix Computations.

A parallel architecture for Kalman filter measurement update and parameter estimation.
Automatica 22, 1 (1986), 783–786.

A new approach to linear filtering and prediction problems.

VLSI Array Processors.

A triangular processor array for computing singular values.

Algorithm engineering—an emerging discipline.

A modified Padé approximation algorithm for matrix manipulation.

Systolic architecture for extended Kalman filtering.
Mi Lu (M'87) received the M.S. and Ph.D. degrees in electrical and computer engineering from Rice University, Houston, TX in 1984 and 1987, respectively. Dr. Lu is an Assistant Professor in the Dept. of Electrical Engineering at Texas A & M University, College Station. Her research interests include parallel computing, distributed processing, parallel computer architectures and applications, array processors, computational geometry, and VLSI algorithms. She is the principal investigator of an NSF grant, and the author of over thirty published papers.

Xiangzhen Qiao received the M.S. degree in computer science from the University of Science and Technology of China in 1981. She was with the Institute of Computing Technology, Academia Sinica, China, where she was an Assistant Professor from 1982 to 1986 and Associate Professor since 1986. Ms. Qiao is currently a research associate in the Dept. of Electrical Engineering, Texas A & M University, College Station. Her research interests include parallel algorithms, performance evaluation of high-performance computers, VLSI algorithms and parallel programming.

Guanrong Chen (M'89) received the M.S. degree in computational mathematics from Zhongshan University, China in 1981, and the Ph.D. degree in mathematics from Texas A & M University, College Station, in 1987. After working as a post-doctorate fellow at the Center for Approximation Theory, Texas A & M University for a few months, and then as a senior research associate and visiting assistant professor in the Dept. of Electrical and Computer Engineering and of Mathematical Sciences at Rice University, Houston, Texas from 1987 to 1990, he joined the faculty of the Dept. of Electrical Engineering, University of Houston, TX in July 1990. His research interests include linear and nonlinear optimal systems control, spline approximation theory and applications, real-time implementation of Kalman filtering, and computational complexity.

Dr. Chen has authored/co-authored three books and over thirty journal papers since 1981.