SN Computer Science (2023) 4:147
https://doi.org/10.1007/s42979-022-01542-x

ORIGINAL RESEARCH

FPGA-Based Hardware Accelerator for Matrix Inversion

Venkata Siva Kumar Kokkiligadda[1] · Vijitha Naikoti[1] · Gaurao Sunil Patkotwar[1] · Samrat L. Sabat[1] · Rangababu Peesapati[2]

Received: 20 April 2022 / Accepted: 2 December 2022 / Published online: 9 January 2023

Abstract
Matrix inversion is a computationally expensive operation in many scientific applications. Performing matrix inversion of rank deficient large order matrices is still a challenge due to its computational overhead. This paper presents the hardware implementation of matrix inversion with (*i*) singular value decomposition (SVD) based on Lanczos and implicit triQR algorithms, (*ii*) *QR* method that uses modified Gram–Schmidt (MGS) technique and (*iii*) inbuilt linear algebra package *QR* inverse using Xilinx Vivado high level synthesis platform. All the three algorithms are implemented on Pynq-Z1 Field Programmable Gate Array (FPGA) using System on Chip (SoC) approach. The resource utilization along with accuracy, hardware execution time with and without loop optimization are reported for the aforementioned matrix inversion techniques of different matrix sizes. We achieved the hardware acceleration factor with loop optimization as 111x and 104x, respectively, for the matrix inversion of the MGS algorithm and SVD based on Lanczos algorithm with respect to the software implementation execution time on Pynq-Z1 FPGA for the matrix size 90×90.

Keywords Matrix inversion · SVD algorithm · Lanczos algorithm · Implicit triQR · *QR* decomposition · Modified Gram–Schmidt

Introduction

Matrix inversion is a fundamental and computational expensive unit in different scientific and engineering applications such as image recognition, least-square problem solving, regression, and multiple-input multiple-output (MIMO) wireless communication. There exist different techniques such as determinant method and decomposition methods for matrix inversion depending on the matrix characteristics. For a non-singular matrix the inverse can be computed easily using Moore-Penrose method i.e., $A^{\dagger} = (A^T A)^{-1} A^T$ [14]. However, it fails to compute the inverse of a singular matrix. Moreover the algorithmic complexity of inverse algorithms increases with the increase in matrix size. Hence, several decomposition algorithms such as lower-upper (*LU*) decomposition, Cholesky decomposition, modified Cholesky decomposition, *QR* decomposition, and singular value decomposition (SVD) are proposed to compute the matrix inverse. From these algorithms, *LU* decomposition can be applied only when the matrix is non-singular square [1], however, it has stability issue. Cholesky decomposition technique can be applied only when the matrix is symmetric, positive

This article is part of the topical collection “Smart and Connected Electronic Systems” guest edited by Amlan Ganguly, Selcuk Kose, Amit M. Joshi and Vineet Sahula.

✉ Rangababu Peesapati
p.rangababu@nitm.ac.in

Venkata Siva Kumar Kokkiligadda
18phpe04@uohyd.ac.in

Vijitha Naikoti
19pimt03@uohyd.ac.in

Gaurao Sunil Patkotwar
19pimt11@uohyd.ac.in

Samrat L. Sabat
slssp@uohyd.ac.in

1 Centre for Advanced Studies in Electronics Science and Technology, University of Hyderabad, Hyderabad, Telangana 500046, India

2 Department of Electronics and Communications Engineering, NIT Meghalaya, Shillong, Meghalaya 793001, India

definite and non-singular square matrix, whereas modified Cholesky can be applied for a negative semi-definite matrix [12]. *QR* decomposition is accurate and stable compared to *LU* decomposition and Cholesky decomposition [6]. For a non-singular matrix, *QR* decomposition can be performed using classical Gram–Schmidt (CGS), Household transformation, or Givens Rotation (GR) techniques. The Givens Rotation method is suitable when the matrix is sparse [2]. The lack of parallelization in Household transformation is the main bottleneck for its embedded hardware implementation [8]. Thus, classical Gram–Schmidt technique is being applied for non-sparse matrix. Although CGS is simpler to implement but it is less stable [13]. Hence, modified Gram–Schmidt is an alternate stable technique being used to compute the inverse of a non-sparse and non-singular matrix. The matrix inversion using *QR* decomposition is applicable only for a full-rank matrix, it fails when the input matrix is an rank-deficient i.e. singular.

Alternatively, singular value decomposition (SVD) is the most efficient decomposition technique in computing the pseudo-inverse of both singular and non-singular matrices. The computational challenging task for computing SVD is the finding of singular values, computed using square root of eigenvalues. The eigenvalues can be obtained using different algorithm techniques. Among these, Jacobi algorithm is more accurate, but it has slow convergence [10]. Jacobi method is simpler technique that transforms a real symmetric matrix to a diagonal matrix using orthogonal transformation, where the elements of diagonal matrix are eigenvalues. For larger matrix, the use of determinant method for computing the coefficient of characteristics polynomial is numerically unstable. The disadvantage of Jacobi method is addressed in Givens rotation method, which transforms the real and symmetric matrix for a tridiagonal matrix and then finds the eigenvalues. Power method is a popular, simple and stable method to find the extreme eigenvalues for a square matrix [7]. However, its convergence depends on the ratio between the two largest eigenvalues and it is not suitable to find all the eigenvalues of a matrix. Golub-Khan algorithm is suitable for finding eigenvalues of a large dense symmetric matrices. Although it has improved computation time but suffers with poor accuracy [5]. Tridiagonalization using the Lanczos algorithm and diagonalization using the *QR* method gives accurate estimates of singular values, reduced computation time and memory storage compared to the above algorithms [9]. Lanczos algorithm generates a sequence of tridiagonal matrices and it is faster because, the input matrix is not varied during the entire process. Lanczos algorithm suffers from loss of orthogonality among Lanczos vectors [3], that need to be compensated using reorthogonalization. The eigenvalues are obtained from the tridiagonal matrix by reducing it to a diagonal matrix. In the current work, we use implicit triQR algorithm to obtain eigenvalues by reducing the tridiagonal matrix to a diagonal matrix using Givens rotation method and Wilkinson shift technique [15], because of its high convergence rate. Thus in this paper, SVD of the matrix is computed based on the Lanczos and implicit triQR algorithm suitable for evaluating the pseudo-inverse of a large real matrix.

FPGAs are becoming popular as computing platforms to implement computational intensive algorithms. In this paper, we implemented the matrix inversion using SVD decomposition based on Lanczos and implicit triQR algorithm with and without inner loop optimization technique on Pynq-Z1 FPGA and compare the hardware acceleration factor, resource utilization with *QR* decomposition based on MGS algorithm and HLS inbuilt *QR* algorithm.

The rest of the paper is organized as follows. "Algorithms for Matrix Inversion" focuses on the matrix inversion algorithms considered for implementation, "Simulation Results" presents numerical simulation results of SVD and *QR* matrix inverse, "Hardware Implementation Results" presents the hardware implementation frame work and performance analysis, followed by conclusions in "Conclusions".

Algorithms for Matrix Inversion

Matrix decomposition algorithms are used to compute the inverse of the matrix for larger dimensions. In the current work, matrix inversion is performed using *QR* decomposition based on modified Gram–Schmidt algorithm and SVD based on lanczos and with implicit triQR algorithm.

Matrix Inversion Using *QR* Decomposition Based on Modified Gram–Schmidt Algorithm

In the *QR* decomposition algorithm, matrix $A \in \Re^{M \times N}$ is decomposed to matrices Q and R such that

$$A = QR, \tag{1}$$

where Q and R represents the orthogonal matrix and upper triangular matrix respectively. The detailed steps for obtaining Q and R using modified Gram–Schmidt is shown in algorithm 1.

Algorithm 1 Calculate $Q, R = qrMGS(A)$

```
Require: $A \in \Re^{M\times N}$
Ensure: $Q, R = qrMGS(A)$
1: for $l = 1 : N$ do
2:     $R(l,l) = \|A(1:M,l)\|_2$
3:     $Q(1:M,l) = A(1:M,l)/R(l,l)$
4:     for $t = l+1 : N$ do
5:         $R(l,t) = Q(1:M,l)^T * A(1:M,t)$
6:         $A(1:M,t) = A(1:M,t) - Q(1:M,l) * R(l,t)$
7:     end for
8: end for
```

After QR decomposition, the inverse of A is obtained as

$$A^{-1} = R^{-1} \times Q^{-1}. \tag{2}$$

Since Q is an orthogonal matrix, it satisfies the orthogonality property; $Q^{-1} = Q^T$, substituting it in Eq. (2), A^{-1} can be rewritten as

$$A^{-1} = R^{-1} \times Q^T. \tag{3}$$

Since $R \in \Re^{N\times N}$ is the upper triangular matrix, R^{-1} can be calculated by inverting the upper triangular matrix as shown in Algorithm 2.

Algorithm 2 Calculate $R^{-1} = inv(R)$

```
Require: $R \in \Re^{N\times N}$
Ensure: $R^{-1} = inv(R)$
1: for $l = 1 : N$ do
2:     for $p = 1 : t-1$ do
3:         $R^{-1}(p,t) = R^{-1}(p,(1:t-1)) * R((1:t-1),t)$
4:     end for
5:     $R^{-1}((1:t-1),t) = R((1:t-1),t)/R(t,t)$
6:     $R^{-1}(t,t) = 1/R(t,t)$
7: end for
```

Matrix Inversion Using SVD Based on Lanczos and Implicit triQR Algorithm

In the SVD decomposition, a real matrix $A \in \Re^{M\times N}$ is decomposed to matrices U, S and V as

$$A_{M\times N} = U_{M\times M} \times S_{M\times N} \times V^T_{N\times N}, \tag{4}$$

where $V \in \Re^{N\times N}$ and $U \in \Re^{M\times M}$ represents the orthogonal matrices and the columns of U and V are the eigenvectors of $A \times A^T$ or $A^T \times A$. The diagonal elements of $S \in \Re^{M\times N}$ are the singular values of the matrix A and remaining elements of S are zero. The singular values are the square root of eigenvalues.

The inverse of a non-singular square matrix $A \in \Re^{N\times N}$ can be computed using SVD as

$$A^{-1}_{N\times N} = V_{N\times N} \times S^{-1}_{N\times N} \times U^T_{N\times N}, \tag{5}$$

where S^{-1} is the inverse of the matrix S. Similarly, the pseudo-inverse of a non-singular rectangular matrix $A \in \Re^{M\times N}$ can be computed using

$$A^{\dagger}_{N\times M} = V_{N\times N} \times S^{\dagger}_{N\times M} \times U^T_{M\times M}, \tag{6}$$

where $S^{\dagger}$ is the pseudo-inverse of the matrix S. The detailed steps for computing pseudo-inverse of the matrix A using SVD based on Lanczos and implicit triQR algorithm is given below.

Initially the input matrix is transformed to a symmetric matrix $B \in \Re^{N\times N}$ as

$$B_{N\times N} = [A^T \times A]_{N\times N}. \tag{7}$$

The Lanczos algorithm is applied on B to get the orthogonal matrix $Q \in \Re^{N\times N}$ and tridiagonal matrix $T \in \Re^{N\times N}$ such that

$$Q^T_{N\times N} \times B_{N\times N} \times Q_{N\times N} = T_{N\times N}. \tag{8}$$

The detailed steps of Lanczos algorithm as shown in Algorithm 3. After obtaining the T and Q matrices, implicit triQR algorithm is applied on T and Q to get the eigenvalues or diagonal matrix $D \in \Re^{N\times N}$ and eigen vector matrix $V \in \Re^{N\times N}$ respectively. The detailed steps of implicit triQR algorithm as shown in Algorithm 4.

In terms of the eigenvalues and eigenvectors, the symmetric matrix B can be expressed as

$$B_{N\times N} = V_{N\times N} \times D_{N\times N} \times V^{-1}_{N\times N}. \tag{9}$$

From Eq. (9), the singular values of matrix A is calculated by applying square root to the diagonal elements of D. The rank of the matrix A can be used to obtain $S \in \mathfrak{R}^{K\times K}$ by taking only singular values greater than the SVD tolerance limit. Since S is a diagonal matrix, and inverse of S i.e $S^{\dagger} \in \mathfrak{R}^{K\times K}$ can be obtained by reciprocating the diagonal elements of S. The eigen vectors $U \in \mathfrak{R}^{M\times K}$ is calculated by considering $k = K$(number of eigenvalues) columns of $V_{N\times N}$.

$$U_{M\times K} = A_{M\times N} \times V_{N\times K} \times S^{\dagger}_{K\times K}. \tag{10}$$

From Eq. (6) the pseudo-inverse $A^{\dagger}$ can be rewritten as

$$A^{\dagger}_{N\times M} = V_{N\times K} \times S^{\dagger}_{K\times K} \times U^{T}_{K\times M}. \tag{11}$$

Substituting Eq. (10) in Eq. (11) gives,

$$A^{\dagger}_{N\times M} = V_{N\times K} \times S^{\dagger}_{K\times K} \times (S^{\dagger}_{K\times K})^{T} \times V^{T}_{N\times K} \times A^{T}_{M\times N}. \tag{12}$$

Since the singular values are square root of eigenvalues, Eq. (12) can be rewritten as

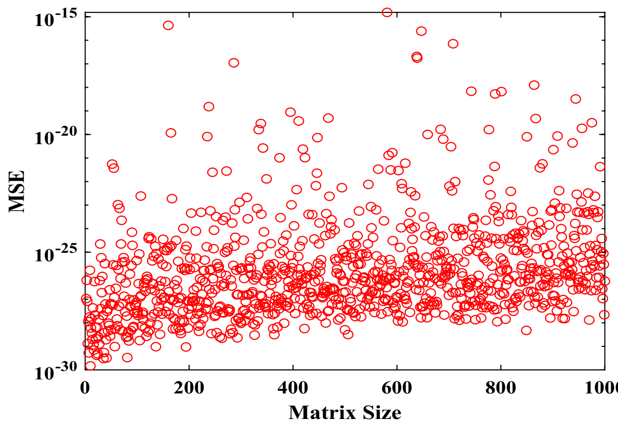

Fig. 2 MSE of matrix inversion using *QR* decomposition based on MGS algorithm

$$A^{\dagger}_{N\times M} = V_{N\times K} \times D_{K\times K} \times V^{T}_{N\times K} \times A^{T}_{M\times N}. \tag{13}$$

The block diagram of matrix inverse computation using SVD based on Lanczos, and implicit triQR algorithm is shown in Fig. 1.

```
Algorithm 3 Calculate $T, Q = Lanczos(B, k)$
Require: $B \in \Re^{N\times N}$, $k$.
Ensure: $T, Q = Lanczos(B, k)$
1: Initialize $i = 0$ , unit vector-$Q_0$, $R_0 = q_1$ , and $\beta_0 = 1$ .
2: while i=0 to $k$ or $\beta_k \neq 0$ do
3:     $Q_{k+1} = R_k/\beta_k$
4:     $i = i + 1$
5:     $\alpha_k = Q_k^T.B.Q_k$
6:     $R_k = (A - \alpha_k.I).Q_k - Q_{k-1}.\beta_{k-1}$
7:     $\beta_k = \|R_k\|$
8: end while
9: Tridiagonal matrix $T$, whose diagonal elements are $\alpha_k$ and sub-diagonal
   elements are $\beta_k$ is obtained. $Q$ is obtained by augmenting column vectors
   $Q_k$.
```

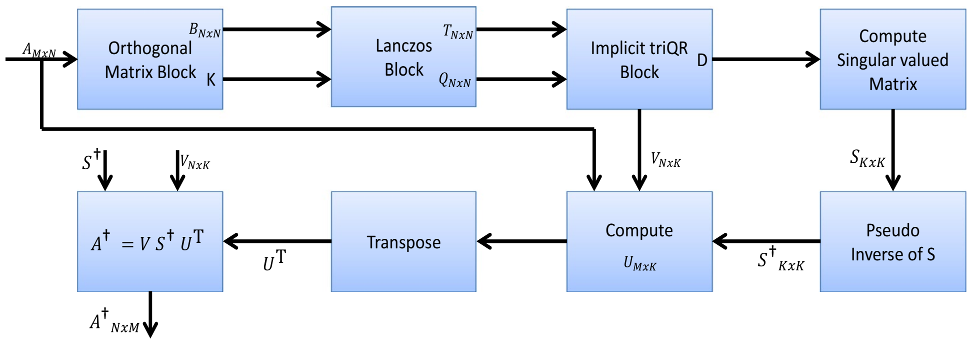

Fig. 1 Block diagram of the matrix inversion using SVD algorithm based on Lanczos, and implicit triQR algorithm

```
Algorithm 4 Calculate $e, V = ImplicitTriqr(T, Q)$
Require: $T \in \Re^{N\times N}$, $Q \in \Re^{N\times N}$.
Ensure: $e, V = ImplicitTriqr(T, Q)$
1: for $m = N : 1$ do
2:     for $i = 0 : convergence$ do
3:         wi = Wilkinson shift ; Gi = Givens rotation
4:         $(T - w_i I) \rightarrow triQR$
5:         $T_{next} = triQR + w_i.I$
6:         $Q_{next} = G_i.Q$
7:         $T = T_{next}$
8:         $Q = Q_{next}$
9:     end for
10: end for
11: After meeting convergence, $T$ is converted to diagonal matrix $D \in \Re^{N\times N}$,
    whose diagonal elements are eigen values $e$ and $Q$ is converted to matrix V.
12: Sorting of eigenvalues $[e, ind]$ = sort $(e$, 'descend' )
13: Update eigen vector matrix $V = Q(:, ind(1 : N))$
```

Simulation Results

The floating-point matrix inversion using (i) *QR* decomposition based on MGS algorithm and (ii) SVD based on Lanczos and implicit triQR algorithm is simulated using Matlab tool to check the functionality. We have considered Matlab inbuilt *pinv* as the reference algorithm. Random input matrices of different sizes are considered as input matrices that need to be inverted. Mean Square Error (MSE) for a matrix of size *t* is obtained as

$$\mathrm{MSE}(t) = \frac{1}{t * t}\sum_{i=0}^{t-1}\sum_{j=0}^{t-1}\left[A_{i,j}^{pinv} - A_{i,j}^{QRinv}\right]^2, \tag{14}$$

where A^{pinv} is the inverse of matrix A using Matlab inbuilt function *pinv* and A^{QRinv} is the inverse of matrix A using MGS based *QR* decomposition algorithm.

Matrix Inversion Using *QR* Decomposition Based on MGS Algorithm

In [13], MGS based algorithm is concluded as the good choice for computing the matrix inverse among the CGS and Householder based inverse technique. In Fig. 2, we applied random input matrix sizes up to 1000×1000 for the MGS based matrix inverse and reported the MSE between the inverse obtained using *QR* decomposition and reference algorithm (*pinv*).

From Fig. 2, it is observed that the mean square error for *QR* inverse based on MGS algorithm is within an error limit of $\leq 10^{-15}$.

Matrix Inversion Using SVD Decomposition Algorithm

We applied SVD based on Lanczos and implicit triQR algorithm to compute pseudo-inverse of both full rank and singular matrices (rank deficient) of different sizes. The MSE

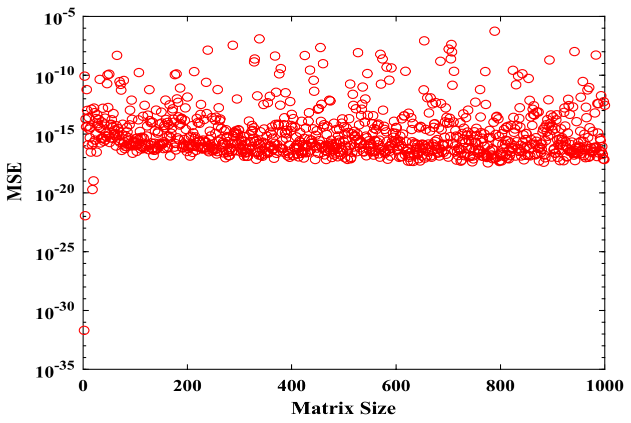

Fig. 3 MSE of matrix inversion using SVD for full rank matrices

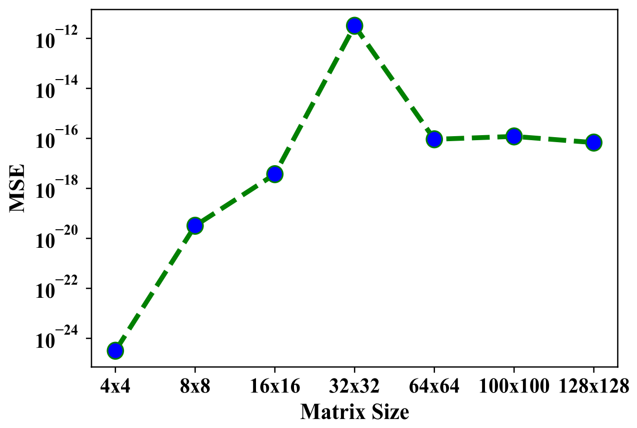

Fig. 4 MSE of matrix inversion using SVD for singular matrices

is obtained same as Eq. (14) except $Qinv(A)$ is replaced with $Sinv(A)$.

where, $Sinv(A)$ is the inverse of matrix A using SVD based on Lanczos and implicit triQR algorithm. MSE of matrix inversion using SVD for full rank and rank deficient matrix are plotted in Figs. 3 and 4 respectively.

From Figs. 3 and 4, it is observed that the mean square error for matrix inversion is within the error limit $MSE \leq 10^{-07}$ verifying the functionality of the inverse using SVD based on Lanczos and implicit triQR algorithm.

Hardware Implementation Results

Matrix inversion using QR decomposition and SVD algorithms are implemented on Pynq-Z1 FPGA. Initially, we tested the functionality of the considered matrix inversion methods using C-simulation on Vivado HLS tool. The clock period is set as 10ns. We synthesized the design with and without inner loop pipelining optimization technique. Further, we created the hardware Intellectual Property (IP) of the matrix inverse algorithms. Overlay is created for this hardware IP block and interfaced with the Zynq IP core using AXI-Lite interface; as shown in Fig. 5. Finally, we realised the design on Pynq-Z1 FPGA with the Jupyter

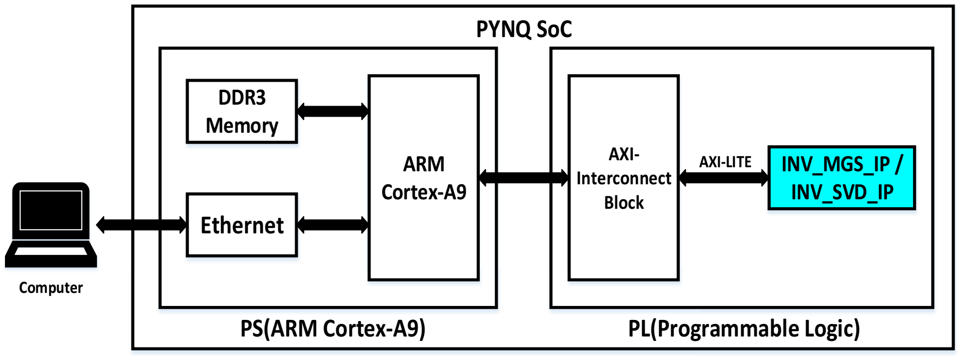

Fig. 5 SoC interface architecture of the Matrix inversion IP

Table 1 Performance analysis of MGS based matrix inversion without inner loop pipelining optimization for various matrix sizes

Matrix size	16×16	25×25	32×32	50×50	90×90
fmax (MHz)	114.29	114.29	114.29	114.29	103.80
Slice LUTs	4387 (8.25%)	4631 (8.70%)	4460 (8.38%)	4620 (8.68%)	4703 (8.84%)
Slice registers	5195 (4.88%)	5527 (5.19%)	5267 (4.95%)	5734 (5.33%)	5771 (5.42%)
BRAM	5 (3.57%)	8 (5.71%)	9 (6.43%)	33 (23.57%)	65 (46.43%)
DSP	5 (2.27%)	6 (2.73%)	5 (2.27%)	5 (2.27%)	5 (2.27%)
LUTRAM	197 (1.13%)	205 (1.18%)	198 (1.14%)	206 (1.18%)	207 (1.19%)
Static power (Watts)	0.137	0.138	0.138	0.141	0.147
Dynamic power (Watts)	1.367	1.374	1.372	1.432	1.506
IP power (Watts)	0.097	0.103	0.101	0.161	0.235
SoC power (Watts)	1.504	1.512	1.51	1.573	1.653
HET (s)	0.0013	0.0045	0.0092	0.0342	0.1965
MSE	$2.59e^{-13}$	$3.17e^{-13}$	$1.61e^{-11}$	$1.97e^{-11}$	$7.41e^{-11}$

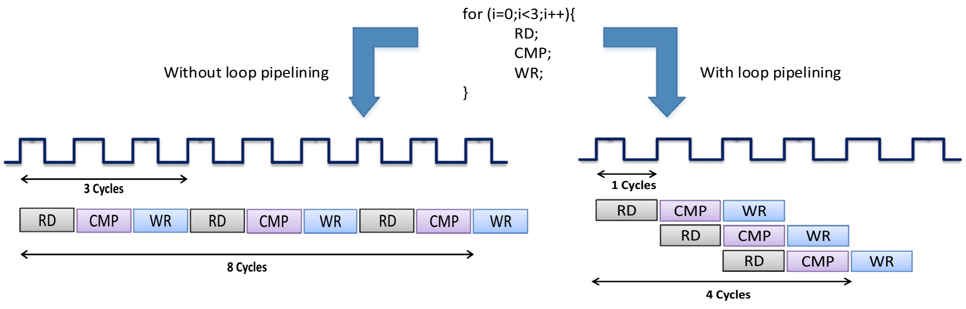

Fig. 6 Illustration of loop pipelining optimization technique [11]

Table 2 Performance analysis of MGS-based matrix inversion with inner loop pipelining optimization for various matrix sizes

Matrix Size	16×16	25×25	32×32	50×50	90×90
fmax (MHz)	109.92	103.80	109.92	103.80	103.80
Slice LUTs	4684 (8.80%)	4928 (9.26%)	4778 (8.98%)	4962 (9.33%)	5124 (9.63%)
Slice registers	5358 (5.04%)	5607 (5.27%)	5450 (5.12%)	5734 (5.39%)	5860 (5.51%)
BRAM	5 (3.57%)	8 (5.71%)	9 (6.43%)	33 (23.57%)	65 (46.43%)
DSP	5 (2.27%)	10 (4.55%)	5 (2.27%)	9 (4.09%)	9 (4.09%)
LUTRAM	210 (1.21%)	223 (1.28%)	213 (1.22%)	228 (1.31%)	230 (1.32%)
Static power (Watts)	0.137	0.138	0.137	0.141	0.146
Dynamic power (Watts)	1.361	1.370	1.362	1.425	1.496
IP power (Watts)	0.091	0.099	0.092	0.155	0.225
SoC power (Watts)	1.499	1.508	1.5	1.567	1.642
HET(s)	0.0008	0.0024	0.0053	0.0181	0.1044
MSE	$2.59e^{-13}$	$3.17e^{-13}$	$1.61e^{-11}$	$1.97e^{-11}$	$7.41e^{-11}$

Table 3 HAF of MGS matrix inverse using without and with inner loop pipelining optimization for different matrix sizes

	SET (s)	WO inner loop pipelining		With inner loop pipelining	
Matrix Size		HET (s)	HAF	HET (s)	HAF
16×16	0.0572	0.0013	44	0.0008	72
25×25	0.2371	0.0045	53	0.0024	98
32×32	0.5157	0.0092	56	0.0053	97
50×50	1.9814	0.0342	58	0.0181	109
90×90	11.6218	0.1965	59	0.1044	111

Notebook using Python. In this paper, we considered single precision floating-point data type for implementation on Pynq FPGA.

Matrix Inversion Using *QR* Decomposition Based on MGS Algorithm

The matrix inversion using *QR* decomposition based on the MGS algorithm is implemented on the Pynq-Z1 FPGA using AXI-Lite interfacing between Processing System (PS) and Programmable Logic (PL) of PYNQ FPGA. We reported the resource utilization, power consumption, hardware execution time (HET) and mean square error for different matrix size in Table 1.

From the Table 1, it is observed that for lower-order matrix MGS method gives less MSE but for the higher-order matrix the accuracy decreases. Similarly the HET increases with increase in matrix dimension.

Pipelining is an optimization technique that decreases the execution time of a function or loop. Figure 6 analyses the

Table 4 Performance analysis of LAP QR based matrix inversion with various matrix sizes

Matrix Size	16×16	25×25	32×32	50×50	90×90
fmax(MHz)	114.29	103.80	114.29	103.80	103.80
Slice LUTs	9934 (18.67%)	10153 (19.08%)	9969 (18.74%)	10186 (19.15%)	10419 (19.58%)
Slice registers	11278 (10.60%)	11588 (10.99%)	11324 (10.64%)	11625 (10.93%)	12270 (11.53%)
BRAM	9 (6.43%)	15 (10.36%)	17 (12.14%)	52 (36.79%)	106 (75.36%)
DSP	34(15.45%)	39 (17.73%)	34 (15.45%)	39 (17.73%)	39 (17.73%)
LUTRAM	776 (4.46%)	830 (4.77%)	732 (4.21%)	844 (4.85%)	674 (3.87%)
Static power (Watts)	0.140	0.141	0.141	0.146	0.154
Dynamic power (Watts)	1.451	1.479	1.454	1.527	1.625
IP power (Watts)	0.183	0.21	0.186	0.259	0.357
SoC power (Watts)	1.591	1.62	1.595	1.673	1.78
HET (s)	0.0026	0.0028	0.0034	0.0072	0.0301
MSE	$1.28e^{-13}$	$1.9e^{-13}$	$4.03e^{-12}$	$5.60e^{-12}$	$2.57e^{-12}$

Table 5 Performance analysis of SVD based matrix inversion for different matrix sizes without inner loop pipelining optimization

Matrix Size	16×16	25×25	32×32	50×50	90×90
fmax (MHz)	108.58	108.75	114.29	114.29	114.29
Slice LUTs	14521 (27.30%)	15326 (28.81%)	14628 (27.50%)	14941 (28.08%)	15081 (28.35%)
Slice registers	16500 (15.51%)	17402 (16.36%)	16587 (15.59%)	17607 (16.55%)	17826 (16.75%)
BRAM	9 (6.43%)	14 (9.64%)	17 (12.14%)	47 (33.21%)	87 (61.79%)
DSP	55 (25%)	55 (25%)	55 (25%)	55 (25%)	55 (25%)
LUTRAM	555 (3.19%)	757 (4.35%)	438 (2.52%)	370 (2.13%)	372 (2.14%)
Static power (Watts)	0.144	0.145	0.145	0.151	0.159
Dynamic power (Watts)	1.622	1.638	1.632	1.709	1.815
IP power (Watts)	0.351	0.367	0.362	0.439	0.544
SoC power (Watts)	1.766	1.783	1.777	1.86	1.973
HET (s)	0.0061	0.0231	0.0462	0.1671	0.9562
MSE	$1.71e^{-11}$	$2.07e^{-11}$	$9.69e^{-08}$	$1.91e^{-07}$	$6.97e^{-08}$

Table 6 Performance analysis of SVD based matrix inversion for various matrix sizes with inner loop pipelining optimization

Matrix size	16×16	25×25	32×32	50×50	90×90
fmax (MHz)	108.75	103.80	114.29	103.80	103.80
Slice LUTs	15699 (29.51%)	16559 (31.13%)	15828 (29.75%)	16179 (30.41%)	16593 (31.19%)
Slice registers	17252 (16.21%)	17940 (16.86%)	17395 (16.35%)	18069 (16.98%)	18374 (17.27%)
BRAM	10 (7.14%)	15 (10.56%)	18 (12.86%)	47 (33.57%)	87 (62.14%)
DSP	55 (25%)	60 (27.27%)	55 (25%)	58 (26.36%)	58 (26.36%)
LUTRAM	540 (3.10%)	708 (4.07%)	443 (2.55%)	386 (2.22%)	389 (2.24%)
Static power (Watts)	0.144	0.145	0.145	0.150	0.157
Dynamic power (Watts)	1.609	1.626	1.628	1.683	1.753
IP power (Watts)	0.339	0.356	0.357	0.413	0.482
SoC power (Watts)	1.754	1.771	1.773	1.833	1.909
HET(s)	0.0037	0.0121	0.0243	0.0932	0.5411
MSE	$1.71e^{-11}$	$2.07e^{-11}$	$9.69e^{-08}$	$1.91e^{-07}$	$6.97e^{-08}$

Table 7 HAF of SVD matrix inverse using without and with inner loop pipelining optimization for different matrix sizes

Matrix size	SET (s)	WO inner loop pipelining		With inner loop pipelining	
		HET (s)	HAF	HET (s)	HAF
16×16	0.3771	0.0061	61	0.0037	102
25×25	1.3672	0.0234	58	0.0121	112
32×32	2.7232	0.0461	59	0.0243	109
50×50	9.7614	0.1679	58	0.0932	104
90×90	56.2591	0.9559	59	0.5411	104

instruction execution without (WO) and with loop pipelining optimization technique. Figure 6 illustrates that, if a task requires eight clock cycles to execute, with enabling loop pipelining technique, the same task can complete the execution in four clock cycles. In this current work, we used pipelining loop optimization method to speed up the hardware execution time.

We applied inner loop pipelining optimization to the MGS-based matrix inversion algorithm. The resources, MSE and HET are tabulated in Table 2. It shows that with inner loop pipelining optimization technique, there is substantial improvement in HET at the cost of negligible change in resources.

We calculated the software execution time (SET) on ARM dual core Cortex-9 processor and used it to calculate the hardware acceleration factor (HAF) as

$$\text{HAF} = \frac{\textsc{Software execution time (SET)}}{\textsc{Hardware execution time (HET)}}. \tag{15}$$

For different matrix size, the HAF with and without inner loop pipelining optimization is tabulated in Table 3. From the Table 3, we observe that, decrease in HET due to inner loop pipelining increases the HAF of the algorithm.

Table 8 HET and FoM comparison of matrix inverse using different algorithm techniques

Matrix size	LAP QR		QR (MGS)		SVD (Lanczos)	
	FoM	HET (s)	FoM	HET (s)	FoM	HET (s)
16×16	170384	0.0026	80496	0.0008	264648	0.0037
25×25	174792	0.0028	84568	0.0024	277192	0.0121
32×32	171160	0.0034	82048	0.0053	266952	0.0243
50×50	175944	0.0072	86240	0.0181	275664	0.0932
90×90	183832	0.0301	89056	0.1044	282056	0.5411

Further, we implemented the matrix inversion using linear algebra package (LAP) *QR* decomposition algorithm available in Vivado HLS tool and report the performance for various matrix sizes in the Table 4.

From the Table 4, it is observed that matrix inversion using LAP *QR* decomposition gives comparable accuracy as MGS-based method, however it consumes more resources compared to the MGS-based inversion method.

Matrix Inversion Using SVD Based on Lanczos and Implicit triQR Algorithms

An IP is created for the matrix inversion using SVD decomposition and implemented on Pynq-Z1 FPGA with AXI-Lite interfacing. We synthesized the design with and without inner loop pipelining optimization technique. The hardware resources, execution time and mean square error for the inversion IP with and without inner loop pipelining optimization for different matrix dimensions is tabulated in Tables 5 and 6 respectively.

SVD based matrix inversion IP gives acceptable functional results.

We evaluated the hardware acceleration factor (HAF) of matrix inversion IP using SVD based on Lanczos and implicit triQR with and without inner loop pipelining optimization. The hardware acceleration factor for matrices of different size are tabulated in the Table 7.

From the Table 7, it is observed that inner loop pipelining optimization gives higher HAF compared to the design without inner loop pipelining optimization irrespective of order of the matrix.

For a fair comparison of resources, we calculate the figure of merit (FoM) parameter as

$$FoM = [(DSP + BRAM) \times 16] + [(FF + LUT) \times 8], \qquad (16)$$

where FF, LUT, DSP, and BRAM represent the number of Flip Flops, Lookup Tables, DSP slices, and Block RAMs used by the SoC. We gave less weight to FF and LUT compared to DSP and BRAM by looking into the available resources. The weight parameter 16 and 8 are chosen empirically. The FoM and HET for all three implementation methods for different matrix sizes are tabulated in Table 8. From the Table 8, it is observed that for lower order matrix, MGS-based *QR* inverse took less hardware execution time and less FoM compared to linear algebra package *QR* inverse, whereas for higher order matrix linear algebra package *QR* inverse took less hardware execution time. FoM of linear algebra package *QR* inverse is nearly double than MGS-based *QR* inverse technique for the higher order matrices. These two methods are applicable for non-singular matrices only. Although SVD based on Lanczos inverse took more HET and FoM compared to linear algebra package *QR* inverse and MGS-based *QR* inverse methods, it is applicable for both singular and non-singular matrices.

Table 9 Performance comparison of different matrix inversion algorithms for matrix size 25×25

Ref	Algo.	fmax (MHz)	Resource Utilization	FoM	Computation time (ms)	Power (mW)	EDP (fJ)	MAE
[4]	GR	NA	BRAM = 10, DSP = 32, FF = 6741, LUT=7143	111744	1.05	124	NA	$2.27e^{-07}$
[4]	Gauss Jordan	NA	BRAM = 3, DSP = 5, FF = 1801, LUT = 2209	32208	3.91	123	NA	$1.56e^{-06}$
Prop. work	MGS	114	BRAM = 8 , DSP = 10, FF = 4928, LUT = 5607	84568	2.4	99	0.0076	$1.79e^{-06}$
[4]	SVD	NA	BRAM = 14, DSP = 49, FF = 9313, LUT = 11340	166232	11.24	126	NA	$3.09e^{-07}$
Prop. work	Lanczos	103.8	BRAM = 15, DSP = 60, FF = 17940, LUT = 16559	277192	12.1	356	0.0331	$1.59e^{-05}$

Table 9 compares resource utilization, FoM, computation time, power consumption, energy delay product (EDP) and maximum absolute error (MAE) of the SVD and MGS-based matrix inverse algorithms for matrix size 25×25.

EDP is computed as

$$EDP = \frac{P}{f_{max}^2}, \tag{17}$$

where P and f_{max} represents total power consumed by the IP and its maximum operating frequency respectively.

The hardware accuracy i.e. maximum absolute error between the input matrix and the reconstructed matrix is computed from the inverse matrix using pseudo inverse property $A_r = AA^{-1}A$ as

$$MAE =| A - A_r |, \tag{18}$$

where A and A_r represents the input matrix and reconstructed matrix evaluated using a pseudo inverse property.

From the Table 9, it is observed that GR and MGS-based matrix inverse algorithm take less FoM, computation time, and power than the proposed SVD-based on Lanczos matrix inverse algorithm. GR and MGS-based matrix inverse algorithm works for only non-singular matrices, but SVD-based on Lanczos matrix inverse works for both singular and non-singular matrices. Gauss Jordan-based matrix inverse algorithm takes less FoM, power, and computation time compared to the proposed SVD-based on Lanczos matrix inverse algorithm, but if the order of matrix increases, then Gauss Jordan-based matrix inverse becomes slow. Similarly, the previously implemented SVD-based matrix inverse algorithm [4] takes less FoM and power, but computation time is almost comparable to the proposed SVD-based on Lanczos matrix inverse algorithm. SVD based on Lanczos is a more appropriate when the input is a large sparse matrix.

Conclusions

In this work, we designed and realized accelerators for matrix inverse algorithms namely (i) QR decomposition based on modified Gram–Schmidt, (ii) SVD based on Lanczos and implicit triQR algorithm, and (iii) linear algebra package QR inverse of Xilinx Vivado HLS on SoC platform in Pynq Jupyter note book environment. After realization, the functionality, area, timing, and power are extracted and compared. We reported the hardware acceleration factor for QR decomposition using MGS-based matrix inversion and SVD based matrix inversion technique. We included reorthogonalization to increase the stability of Lanczos algorithm. The proposed SVD based matrix inverse is the most efficient decomposition technique in computing the pseudo-inverse of a square and rectangular rank deficient matrix. In future, we will develop a fixed-point matrix inversion optimized hardware IP of the proposed matrix inversion algorithms to reduce hardware resources.

Acknowledgements The first author would like to thank the University Grants Commission, Govt. of India for providing research fellowship to carryout this work.

Declarations

Conflict of Interest The authors declare that they have no conflict of interest.

References

1. Baliarsingh P, Nayak L, Kumar V. On matrix inversions through difference operators. Iranian J Sci Tech Trans A: Sci. 2018;42(4):2069–77.
2. Chang RCH, Chih-Hung L, Kuang-Hao L, et al. Iterative QR decomposition architecture using the modified Gram-Schmidt algorithm for MIMO systems. IEEE Trans Circuits Syst I: Regular Papers. 2010;57(5):1095–102.
3. Chen J, Saad Y. Lanczos vectors versus singular vectors for effective dimension reduction. IEEE Trans Knowl Data Eng. 2008;21(8):1091–103.
4. Chetan S, Manikandan J, Lekshmi V, et al. Hardware implementation of floating point matrix inversion modules on FPGAs. In: Proc. IEEE 32nd International Conference on Microelectronics (ICM), 1–4 , 2020.
5. Chung J, Saibaba AK, Brown M, et al. Efficient generalized Golub-Kahan based methods for dynamic inverse problems. Inverse Problems. 2018;34(2): 021005.
6. Kokkinos Y, Margaritis KG. Managing the computational cost of model selection and cross-validation in extreme learning machines via Cholesky, SVD, QR and eigen decompositions. Iranian J Sci Tech Trans A: Sci. 2018;295:29–45.
7. Nigam N, Pollock S. A simple extrapolation method for clustered eigenvalues. Num Algorithms. 2022;89(1):115–43.
8. Omran SS, Abdul-abbas AK. Design and implementation of 32-bits MIPS processor to perform QRD based on FPGA. In: Proc. IEEE International Conference on Engineering Technology and their Applications (IICETA), 36–41, 2018.
9. Pradhan T, Routray A, Kabi B. Comparative evaluation of symmetric SVD algorithms for real-time face and eye tracking. In: Matrix information geometry. Springer, pp. 323–40, 2013.
10. Shi Z, He Q, Liu Y. Accelerating parallel Jacobi method for matrix eigenvalue computation in DOA estimation algorithm. IEEE Trans Vehicular Tech. 2020;69(6):6275–85.
11. Solod P, Jindapetch N, Sengchuai K, et al. High level synthesis optimizations of road lane detection development on ZYNQ-7000. Pertanika J Sci Tech. 2021. 10.47836/pjst.29.2.01.
12. Venkata Reddy K, Rangababu P, Sabat SL. System on chip implementation of low complex orthogonal matching pursuit algorithm on FPGA. In: Proc. 2020 IEEE 6th International Conference on Signal Processing and Communication (ICSC), pp. 178–84, 2020.
13. Venkata Siva Kumar K, Venkata Reddy K, Sabat SL, et al. System on chip implementation of floating point matrix inversion

using modified Gram-Schmidt based QR decomposition on PYNQ FPGA. In: Proc. IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS), pp. 84–8, 2021.

14. Yang Q, Xiaoji L, Yaoming Y. Developing reverse order law for the Moore-Penrose inverse with the product of three linear operators. J Math. 2021;2021:6585951.
15. Zee FGV, de Geijn RAV, QuintanaOrt G. Restructuring the tridiagonal and bidiagonal QR algorithms for performance. ACM Trans Math Softw (TOMS). 2014;40(3):1–34.