Design And Implementation Of A Parallel Pipelined Matrix Transposition Architecture Using Shift Registers

Gangulappagari Usha Sree[1] , Mr. B.C. Vengamuni,M.Tech, (Ph.D)[2] , Dr. M Madhu Babu[3]

[1]*M.Tech, Student, Department of ECE, JNTUCEA.ushasreeg2002@gmail.com*
[2]*Assistant Professor, Adhoc Department of ECE, JNTUACEA. vengamuni.phd@gmail.com*
[3]*Assistant Professor, Adhoc Department of ECE, JNTUACEA. madhu07vlsi@gmail.com*

Keywords: Matrix Transposition, Pipelined Architecture, Memory, Latency, Processing elements

ABSTRACT:

An efficient algorithm and architecture for matrix transposition using registers is proposed, enabling parallel processing with reduced latency and complexity. The design supports K-parallel transposition, where K represents the level of parallelism, and achieves minimal latency and memory usage. This architecture employs a sequence of uniform swap units arranged in a cascaded manner, where the activation of each stage is algorithmically defined and driven by counter-based control logic. The design supports matrix transposition for dimensions that are integer multiples of K, where K is not constrained to powers of two. As part of this study, a 3-parallel and a 4-parallel architecture are implemented for transposing a 12×24 matrix. A performance comparison shows that the 4-parallel architecture performs better in terms of processing efficiency and resource utilization. The results also offer deeper insights into continuous-flow transposition for non-square matrices.

1. INTRODUCTION:

Matrix transposition is a fundamental operation essential to numerous domains, including image signal processing [1], artificial intelligence [2], and various engineering applications [3], [4]. It is a key element in matrix-based computations and is widely applied in deep learning models, especially in computer vision [5] and natural language processing [6]. In applications such as image compression and synthetic aperture radar (SAR) imaging [7], matrix transposition is crucial for executing two-dimensional (2D) fast Fourier transforms (FFTs), typically implemented through successive one-dimensional (1D) FFTs. It is essential in neural networks, notably in dense layers, convolutions, and attention modules. Traditionally, zero-padding has been employed to reshape input data into square or rectangular formats, ensuring structural consistency during the training phase [2].

Wang's design is restricted to transposing square matrices, while Garrido and colleagues [8] developed a register-based method that optimizes both memory consumption and latency—but it only applies to matrices whose dimensions are powers of two. Later, Garrido and Pirsch [9] expanded this approach to handle non-square matrices using memory-centric designs; however, these implementations do not meet the theoretical minimum requirements for memory usage. In contrast, Zhang et al. [10] proposed an architecture capable of reaching the theoretical lower bounds for both memory and latency in non-square matrix transpositions.

Zhang's architecture is specifically designed for matrices where one dimension is an exact multiple of the other. This restricts its applicability, as scientific computing often involves matrices with arbitrary sizes that do not conform to fixed aspect ratios [11].

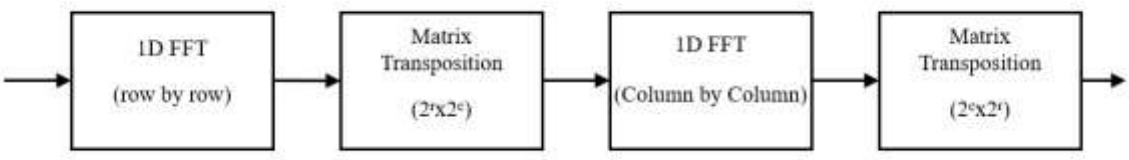

Figure 1: Procedure for Computing 2D FFT Using Continuous-Flow Architecture

To overcome these constraints, this paper presents a novel continuous-flow algorithm and corresponding hardware architecture designed to transpose matrices where both dimensions are integer multiples of a specified parallelism factor, K. The proposed approach achieves the theoretical lower bounds for latency and memory usage. Section 2 introduces the foundational algorithm for 3-Parallel Architecture, while Section 3 focuses on the the foundational algorithm for 4-Parallel Architecture. A detailed performance comparison with existing approaches is presented in Section 4, followed by concluding observations in Section 5.

2. BASIC PIPELINED ALGORITHM FOR 3-PARALLEL ARCHITECTURE

Matrix transposition is a core operation in linear algebra, formally expressed as:

$M_{i,j} = (M^T)_{j,i}$

where i = 0, 1, 2, ..., R − 1 and j = 0, 1, 2, ...,C − 1. Here, A and B denote the number of rows and columns in the matrix, respectively.

To simplify the description in the subsequent sections, we define: $A = R / K, \quad B = C / K$

where K denotes the level of parallelism applied in the architecture. Algorithm 1 outlines a structured Matrix transposition requires K to be a common divisor of the matrix dimensions, enabling structured data exchange across three algorithmic steps: within arrays (Step 1), within blocks (Step 2), and between arrays (Step 3). Each step comprises cascaded stages, and use defined swap operations with calculated offsets and positions. For example, transposing a 12×24 matrix with K = 3 involves two stages in Steps 1 and 2, and ten in Step 3, completing the full data rearrangement process. [12].

Algorithm 1: K-Parallel Matrix Transposition (K=3)

Input: R(rows), C(columns), K(Parallelism), and matrix M
Output: Transposed matrix M^T
Phase 1: Rearranging elements inside each array
1. Loop over each array index a from 0 to A − 1:
2. Loop through s1 = 0 to (K − 1) × (A − 1) – 1:
3. Compute range start and range size using:
* row_start = [K − 1 − mod(s1, K − 1)][B − int(s1, K − 1)]
* row_count = [mod(s1, K − 1) + 1][B − 1 − int(stage1, K − 1)]
4. From r = row_start − 1 to row_start − 1 + row_count − 1
5. For each column c from 0 to K – 1:
6. Swap elements (M[r, c], M[r + 1, c]) using PEII
Phase 2: Reordering within blocks
7. Iterate over every block index 0 to A × B – 1:
8. For s2 from 0 to K – 2:
9. Loop r from 0 to K − 2 − s2
10. Loop c from s2 + 1 to K − 1
11. Swap (M[r, c] with M[r + 1, c − 1]) possessed by PEI
Phase 3: Itransferring elements across arrays
12. For each stage index s3 in 0 to (A − 1) × (BK − 1) − 1
13. Determine the dynamic start and count of rows:
* row_start_1 = [A − 1 − mod(s3, A − 1)][BK − int(s3, A − 1)]

* row_count_1 = [mod(s3, A − 1) + 1][BK − 1 − int(s3, A − 1)]
14. From r = row_start_1 – 1 to row_start_1 – 1 + row_count_1 −1
15. Loop over c from 0 to K − 1
16. Swap M[r, c] with M[r + 1, c]) using PEII unit

Return:

* Final transposed matrix M^T

2.1 PERFORMANCE OF CASCADED 3-PARALLEL ARCHITRCTURE:

Transposing a R×C matrix is carried out through multiple permutation stages, each governed by the logic of the core algorithm.

A single-path swap circuit, as described in [14], enables the exchange of two data points separated by a defined offset. The swap or pass-through operation within these circuits is controlled by signals that dynamically configure the internal multiplexers.

The cascaded K-parallel matrix transposition architecture comprises two types of processing elements: PEI, which handles data swaps across input ports (Step 2 of Algorithm 1), and PEII, which manages intra-port swaps across clock cycles (Steps 1 and 2). The number of stages in each step is derived from Algorithm 1 based on matrix dimensions (R, C) and parallelism K. Control is managed by three counters (C_0, C_1, C_2), each generating stage-specific signals (Sn) for different transposition steps.

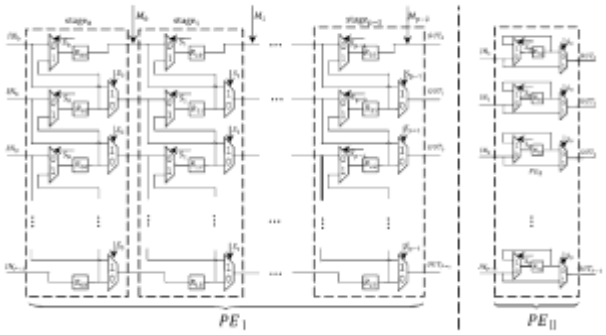

Figure 2: K-Paralle PE's for Algorithm 1

2.2 CONTRL STRATEGY:

The operation of the proposed architecture is governed by a set of counters that manage control signals across different stages. The control signal corresponding to the n-th stage in Steps 1, 2, and 3 is represented as S_n These signals are derived from three dedicated counters C_0, C_1, and C_2 which count from 0 up to BK−1, K−1 and ABK−1, respectively. Each counter increments its value on every clock cycle to ensure proper sequencing and timing across the architecture.

For an R × C matrix, the number of stage1 in step 1 is (A −1)×(K−1); the number of stage2 in step 2 is K − 1; the number of stage3 in step 3 is (BK−1)×(A −1); so, the total size of the memory is as follows:

$$D = (B - 1) \times (K - 1) \times K + (K - 1) \times K + (BK - 1) \times (A - 1) \times K,$$

$$= (BK - 1)(AK - 1) + K - 1,$$

$$= (C - 1)(R - 1) + K - 1$$

and the total latency is as follows:

$$L = D/P,$$

$$= [(C - 1)(R - 1) + K - 1]/K.$$

As discussed in [15, Sec. 3.3] and [16], the K-path architecture achieves the theoretical minimum requirements for both memory consumption and processing latency. Figures 3 and 4 illustrate the simulation outcomes and the architectural design of the 3-parallel implementation for a 12×24 matrix, respectively.

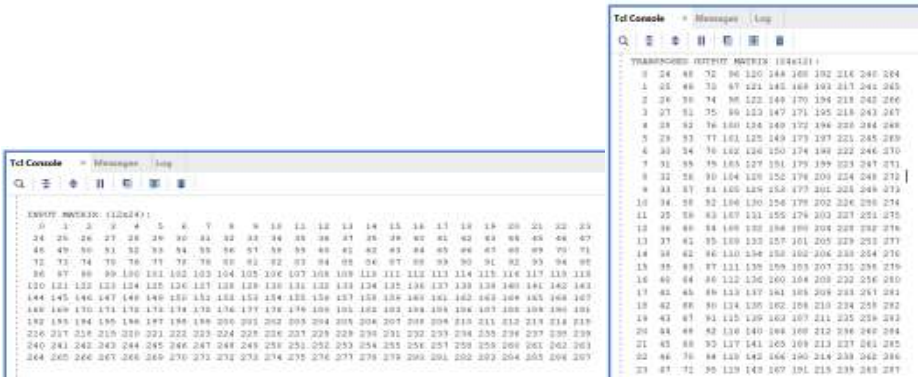

Figure 3: Simulation results of 3-Parallel Architecture for 12x24 matrix

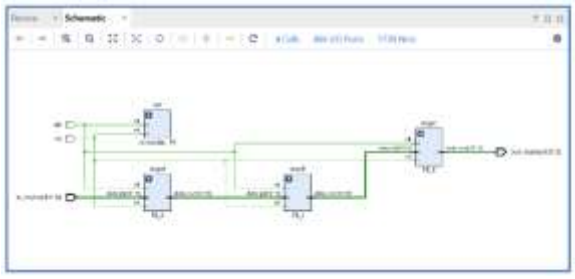

Figure 4: Schematic resukts of 3-parallel Architecture for 12x24 matrix

3. BASIC PIPELINED ALGORITHM FOR 3-PARALLEL ARCHITECTURE:

A = min(R;C) ; B = max(R;C);

Q=B/A

Algorithm 2: K-Parallel Matrix Transposition (K = 4)
Input: R, C and matrix M **Output:** matrix M^T **Phase 1:** Internal element shifting within a block 1. For each pipeline step s from 0 to A − 1 2. Repeat for every processing segment p from 1 to Q: 3. Compute start index: base_row = (p − 1)A 4. Iterate over row indices r from base_row to base_row + pA – 2 – s 5. Loop over columns c ranging from s + 1 to A − 1

6. Swap the pair of entries (M[row, l], M[row + 1, c])
Phase 2: Inter-block data exchanges
7. For each processing cycle s from 0 to (Q -1)(A – 1) - 1 :
8. Compute:
* base_row = [Q − 1 − mod(s, Q − 1)][A − int(s1, Q − 1)]
* shift = [mod(s, K − 1) + 1][A − 1 − int(s1, Q − 1)]
9. From base_row – 1 up to base_row – 1 + shift – 1
10. For each column index c from 0 to A – 1
11. Swap M[row, c], M[row + 1, c − 1]
Return:
M^T, the transpose of matrix M

Based on the stage-wise depiction in Algorithm 2, Steps A and B are amenable to pipelining within hardware circuits. The overall transposition architecture remains structurally consistent for both C<R and C>R, with minor variations in control logic and execution sequence. Specifically, Step A precedes Step B when R>C, while the order is reversed for R<C; the architecture and control strategies for the R>C case are detailed in Section 3.2, and the alternative scenario follows a similar design.

3.1 PERFORMANCE OF CASCADED 4-PARALLEL ARCHITRCTURE:

The proposed R×C matrix transposition architecture is composed of two main parts: A−1 cascaded basic permutation units, and an additional (K−1)(A−1) cascaded stages, each incorporating a shift register of length A. In total, the architecture integrates K(A-1) basic permutation units arranged in cascade.

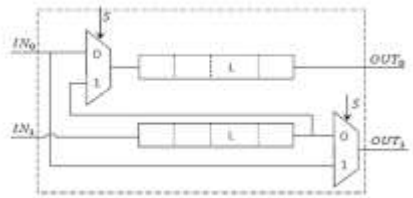

Figure 5: A two-path basic exchange circuit utilizing shift registers of length L

3.2 CONTROL STRATEGY:

We introduce a multi-path permutation circuit tailored for non-square matrices by building upon the foundational exchange circuit proposed by Cheng and Yu [14], as well as the N-parallel permutation architecture developed by Wang [15]. While the overall control approach remains consistent with that described in Section 2.2, the control signal generation is simplified. Specifically, the counter C_0 is no longer necessary in the multi-path configuration. As a result, control signals are now driven by just two counters: C_1 and C_2, which count from 0 to A−1 and 0 to QA−1, respectively.

the total memory size is as follows:

$$D = (A-1) \times A + (Q-1)(A-1) \times A$$
$$= (R-1)(C-1)+A-1$$

and the total latency is as follows:

$$L = D/P,$$

$$= [(C-1)(R-1) + K - 1]/K$$

Consequently, validation results confirm that the design attains the theoretical minimums in both latency and memory usage. Figures 6 and 7 illustrate the simulation outcomes and the architectural design of the 4-parallel implementation for a 12×24 matrix, respectively.

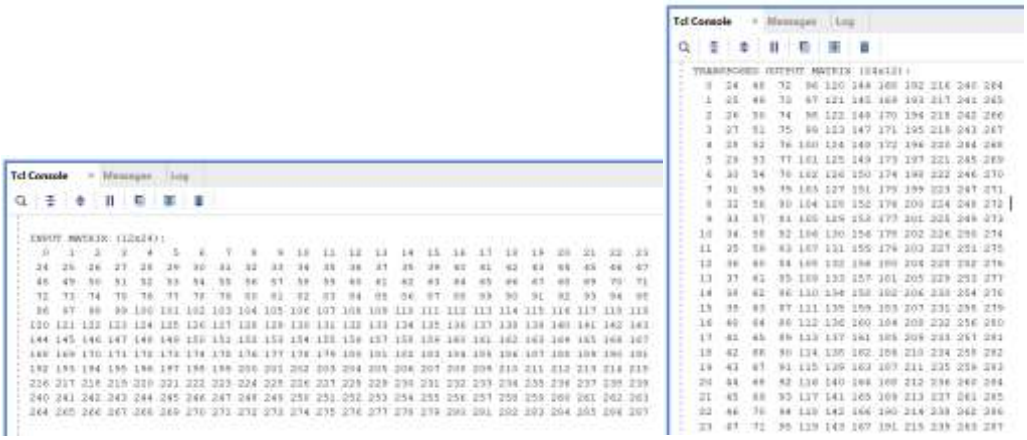

Figure 6: Simulation results of 4-Parallel Architecture for 12x24 matrix

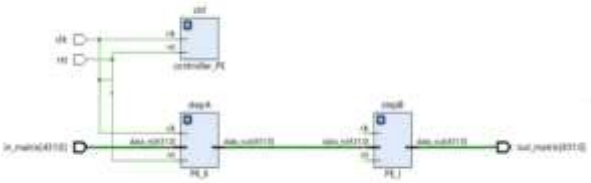

Figure 7:Schematic results of 4-Parallel Architecture for 12x24 matrix

4. COMPARISION AND RESULTS:

MATRIX	K	MEMORY	LATENCY
12X24	3	$(C-1)(R-1)+K-1$	$[(C-1)(R-1)+K-1]/K$
12X24	4	$(R-1)(C-1)+A-1$	$[(C-1)(R-1)+A-1]/K$

This section presents a comparative analysis of the proposed matrix transposition architecture against several leading designs. The architecture described in [16] supports both serial and parallel transposition for square matrices of size $2^n \times 2^n$, achieving optimal memory and latency performance. Its multiplexer complexity is $O(N\log_2 N)$. Similarly, the approach in [15] also reaches these theoretical bounds for N×N square matrices but incurs higher multiplexer complexity, scaling as $O(N^2)$. Notably, both of these designs are limited to square matrix configurations and cannot be directly applied to non-square matrices.

In contrast, the design presented in [10] extends functionality to non-square matrices and lowers multiplexer complexity back to $O(N\log_2 N)$; however, this comes at the cost of increased memory and latency, both growing as MN for an M×N matrix—well above the theoretical minimum.

The architecture proposed in this work addresses these limitations by supporting both square and non-square matrices, including those with row and column dimensions that are integer multiples. Although it introduces a slightly

higher multiplexer complexity of $O(N^2)$, it attains the theoretical lower bounds for both memory and latency, requiring only $(M-1)(N-1)$ registers and an equal number of clock cycles. Furthermore, the design supports scalability across serial and K-parallel implementations, enhancing its applicability to a wide range of use cases.

To assess performance and hardware efficiency, the architecture was implemented on a Xilinx Zynq-7000 FPGA (xc7z020clg484-1) using the Vivado design suite. For consistency, Garrido's architecture was also implemented using equivalent register-based memory to ensure a fair comparison.

Compared to the solution in [10], While the proposed architecture incurs a modest increase in hardware resource usage due to additional multiplexers, it achieves the theoretical minimum in latency and maintains competitive throughput. Moreover, it provides greater flexibility by efficiently supporting a wider range of matrix dimensions, including those not limited to fixed-size multiples. Power consumption is predominantly driven by flip-flops (FFs) and look-up tables (LUTs) within the transposition modules, demonstrating an effective trade-off between performance and hardware cost.

Theoretical minimums	**12x24 (3-Parallel)**	**12x24 (4-Parallel)**
Memory	255	265
Latency	85	64

5. CONCLUSION:

This paper introduces an efficient matrix transposition architecture using shift registers, optimized for K-parallel iplementations. By leveraging cascaded swap units and a counter-based control mechanism, the design achieves minimal latency and memory usage while supporting a broad range of non-square matrices with dimensions that are integer multiples of K. The implemented 3-parallel and 4-parallel transposition architectures for a 12×24 matrix demonstrate the scalability and effectiveness of the proposed approach. Notably, the 4-parallel design achieves improved processing efficiency and better resource utilization, validating the practicality of the architecture for high-performance applications in digital signal processing and beyond.

REFERENCES:

[1] F. B. a. F. L. M. Bian, "Matrix transpose methods for SAR imaging system," in Proc. IEEE 10th Int. Conf. Signal Process, Beijing, China, Oct. 2010.

[2] D. H. S. C. S. K. a. H.-J. Y. D. Im, "DT-CNN: An energy efficient dilated and transposed convolutional neural network processor for region of interest based image segmentation," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. Oct. 2020, p. 3471–3483, 2020.

[3] M. T. L.-G. Ö. a. U. S. F. Mahmood, "2D discrete Fourier transform with simultaneous edge artifact removal for realtime applications," in Proc. Int. Conf. Field Program. Technol. (FPT), Queenstown, New Zealand, 2015.

[4] S. M. a. K. Jayakumar, "A DSP based real-time 3D FFT system for analysis of dynamic parameters," in Proc. IEEE Int. Conf. Adv. Commun. Control Comput. Technol.,, Ramanathapuram, India, 2014.

[5] H. Song, "The application of computer vision in responding to the emergencies of autonomous driving," in Proc. Int. Conf. Comput. Vis. Image Deep Learn. (CVIDL), Chongqing, China, 2020.

[6] A. Barnard, "The nursing profession: Implications for AI and natural language processing," in Proc. Int. Conf. Natural Lang. Process. Knowl. Eng, Beijing, China, 2007.

[7] C. C. C. D. D. S. F. P. Q. Q. F. C. X. W. a. X. Y. Z. K. Han, "An accurate 2D nonuniform fast Fourier transform method applied to high resolution SAR image reconstruction," in 2012 International Workshop on Metamaterials (Meta). IEEE, 2012 .

[8] J. G. a. O. G. M. Garrido, "Optimum circuits for bitdimension permutations," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 5, no. May 2019, p. 1148–1160, 2019.

[9] M. G. a. P. Pirsch, "Continuous-flow matrix transposition using memories," IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 67, no. Sep. 2020., p. 3035–3046, 2020.

[10] Z. M. a. F. Y. B. Zhang, "A novel pipelined algorithm and modular architecture for non-square matrix transposition," IEEE Trans. Circuits Syst. II, Exp. Briefs, Vols. 68, no. 4, no. Apr. 2021, p. 1423–1427, 2021..

[11] Z. Q. M. N. a. W. Y. S. Yang, "TransPose: Keypoint localization via transformer," in Proc. IEEE/CVF Int. Conf. Comput. Vis., 2021, 2021.

[12] Z. M. ,. a. W. L. Bo Zhang, "Parallel Pipelined Architecture and Algorithm for Matrix Transposition Using Registers," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: EXPRESS BRIEFS, Vols. 69, no. 3, no. March 2022, pp. 1627-1631, 2022.

[13] J. G. a. O. G. M. Garrido, "Optimum circuits for bit reversal," IEEE Trans. Circuits Syst. II, Exp. Briefs, Vols. 58, no. 10, no. Oct. 2011, p. 657–661, 2011.

[14] C. C. a. F. Yu, "An optimum architecture for continuous-flow parallel bit reversal," IEEE Signal Processing Letters, Vols. vol. 22, no. 12, no. 2015, p. 2334–2338, 2015.

[15] Z. M. a. F. Y. Y. Wang, "Pipelined algorithm and modular architecture for matrix transposition," IEEE Transactions on Circuits and Systems II: Express Briefs, Vols. vol. 66, no. 4, no. Apr. 2019, p. 652–656, 2019.

[16] P. S. H. S. a. J. T. T. Järvinen, "Stride permutation networks for array processors," J. VLSI Signal Process. Syst. Signal Image Video Technol., Vols. vol. 49, no. 1, no. 2007, p. 51–71, 2007.