

Article

Ordered Eigenvalue Decomposition Implementation on Systolic Arrays via Virtual Rewiring

Chengqian Tang ¹, Yaxuan Lu ¹, Bowen Liang ¹, Yunhe Cao ^{1,*} and Mengmeng Han ²

¹ National Key Laboratory of Radar Signal Processing, School of Electronic Engineering, Xidian University, Xi'an 710071, China; 23021211242@stu.xidian.edu.cn (C.T.); 24181214092@stu.xidian.edu.cn (Y.L.); 23021211244@stu.xidian.edu.cn (B.L.)

² Beijing Electro-Mechanical Engineering Institute, Beijing 100076, China; hanmengmeng666@163.com

* Correspondence: cyh_xidian@163.com

Abstract

Eigenvalue Decomposition (EVD) is a fundamental operation in real-time signal processing, yet obtaining sorted outputs from systolic arrays remains a persistent engineering challenge. The conventional Brent–Luk architecture relies on external sorting networks to reorder eigenvalues. Attempts to achieve in-place sorting via angle adjustment fail due to “topological mismatch,” a conflict between implicit data permutation from large-angle Givens rotations and fixed hardware routing. To address this, we propose a virtual rewiring mechanism. By exploiting the inherent half-cycle reversal pattern of Round-Robin scheduling, we derive a correction algorithm requiring only sign-bit operations. This achieves automatic descending-order arrangement without modifying physical interconnects. Field-Programmable Gate Array (FPGA) experiments demonstrate that the proposed scheme requires negligible additional resources (0.8% Look-Up Tables (LUTs)) while reducing sorting-related logic by 91%. Furthermore, sorting is achieved entirely within the existing computational pipeline, resulting in zero additional hardware latency per sweep.

Keywords: EVD; systolic array; Brent–Luk architecture; topological mismatch; virtual rewiring; FPGA

1. Introduction

Eigenvalue Decomposition (EVD) underpins a broad spectrum of core applications in modern signal processing. Multiple Input Multiple Output (MIMO) precoding extracts dominant channel directions to achieve optimal beamforming [1–3]. Subspace tracking algorithms leverage EVD to separate adaptive signals in dynamic environments [4]. Principal component analysis relies on EVD to accomplish dimensionality reduction within machine learning pipelines [5]. As these applications migrate toward edge devices, microsecond-level latency constraints impose stringent demands on hardware EVD implementations [6].

Systolic arrays, by virtue of their inherent parallelism and numerical stability, have emerged as the dominant architecture for hardware EVD [7]. The seminal scheme proposed by Brent and Luk organizes $n/2$ processing elements (PEs) into a linear array, executing Givens rotations in parallel under Round-Robin scheduling to achieve $O(n)$ parallelism [8]. A fundamental deficiency, however, has long been overlooked. The Jacobi method outputs eigenvalues in an order dictated by the input matrix structure rather than by magnitude. For applications requiring the extraction of the top- k principal components, unordered output necessitates an appended sorting stage—existing solutions cascade an independent



Academic Editor: Alexander Barkalov

Received: 11 January 2026

Revised: 9 February 2026

Accepted: 14 February 2026

Published: 25 February 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

sorting network after the EVD core, yet the resulting resource overhead and latency penalty are frequently underestimated.

An intuitive alternative arises naturally, as follows: can rotation angles be adjusted within the Jacobi iteration such that larger eigenvalues consistently occupy smaller indices, thereby accomplishing sorting “incidentally” during computation? Mathematically, replacing $\arctan$ with an appropriately configured $\arctan2$ suffices to guarantee $a'_{pp} \geq a'_{qq}$ after each rotation; the 0–1 principle from sorting network theory [9] suggests convergence under idealized conditions, and Monte Carlo validation (10^6 trials) confirms that four Jacobi sweeps suffice for arbitrary inputs. This seemingly elegant scheme, however, faces a fundamental obstacle in Brent–Luk systolic arrays.

The crux lies in what we define as topological mismatch as follows: when the rotation angle exceeds $\pm\pi/4$, the Givens transformation implicitly executes a data exchange between indices p and q . At the abstract algorithmic level, this permutation is transparent and harmless, yet the dataflow paths of a systolic array are physically fixed at design time by the Round-Robin schedule—the global routing operator Π is hardwired, whereas the local permutation π_{local} depends on runtime data, rendering the two irreconcilable. The consequences are critical, as follows: matrix $\mathbf{A}$ evolves along the actual path $\Pi \circ \pi_{\text{local}}$, while the eigenvector matrix $\mathbf{V}$ continues to update along the intended path Π . The divergence of these two trajectories invalidates the eigenvalue–eigenvector correspondence. Prior work either sidesteps this issue by accepting unordered output [8,10], or it incurs the area and latency overhead of external sorting [11,12]. In-place sorting within a systolic array thus remains an open challenge.

This paper fundamentally reframes the problem as follows: rather than fighting against fixed interconnects, we exploit the deterministic structure of Round-Robin scheduling to predict and proactively correct phase conflicts. The central discovery is that index reversal is not a random perturbation but follows a precise half-cycle pattern—during the time-step interval $n/2 < t < n - 1$, all non-pivot PEs systematically undergo index inversion. This topological invariant admits algebraic compensation. The ostensibly complex permutation correction ultimately collapses to a single-bit sign operation on the rotation factor, equivalent to accomplishing “virtual rewiring” without touching physical interconnects.

The contributions of this paper are as follows:

1. We reveal the root cause of in-place sorting failure in systolic arrays—the conflict between implicit permutation and fixed routing—and formalize it as the topological mismatch problem.
2. We derive the half-cycle reversal law governing index-phase evolution under Round-Robin scheduling and establish a complete phase classification system for processing elements.
3. We propose a virtual rewiring mechanism requiring only sign-bit operations, enabling descending-order sorting to be embedded directly into existing Brent–Luk architectures without structural modification.
4. Field-Programmable Gate Array (FPGA) implementation demonstrates that the proposed scheme incurs merely 0.8% additional Look-Up Table (LUT) resources while eliminating 91% of the logic overhead of external sorting networks, achieving per-sweep zero additional latency and convergence accuracy on the order of 10^{-4} .

The remainder of this paper is organized as follows: Section 2 surveys related work on hardware sorting networks and systolic array-based EVD implementations. Section 3 reviews the mathematical foundations of Jacobi EVD and the Brent–Luk systolic array. Section 4 analyzes the descending-order strategy, establishes its theoretical foundation through the implicit sorting network perspective, and characterizes the topological mismatch phenomenon it induces. Section 5 derives the phase correction algorithm, including

the half-cycle reversal law, PE phase classification, and unified correction logic. Section 6 describes the hardware architecture with emphasis on the decoupled parallel update mechanism for eigenvectors. Section 7 presents experimental results covering resource consumption, latency, convergence accuracy, and scalability analysis. Section 8 discusses the routing bottleneck and scalability advantages of the proposed approach. Section 9 concludes the paper.

2. Analysis of Related Work

This section surveys the literature along two parallel threads—hardware sorting networks and systolic array-based EVD implementations—to expose the shared predicament that existing approaches face when ordered eigenvalue output is required.

2.1. Hardware Sorting Networks

The design of hardware sorting networks has long constituted a research focus in Very Large Scale Integration (VLSI) [12]. Batcher’s bitonic sorter cascades compare–exchange units to achieve deterministic ordering. Its $O(\log^2 N)$ depth and $O(N \log^2 N)$ comparator count have established it as the canonical choice for FPGA deployment [11,13]. Peters et al. restructured the memory access patterns of the bitonic network for Graphics Processing Unit (GPU) architectures [13], while Mueller et al. systematically explored the parallelism-resource trade-off of sorting networks on FPGAs [14]. The structural regularity of such schemes facilitates pipelining, yet it imposes the following rigid penalty: comparator count scales tightly with data size, precluding elastic tailoring to application-specific requirements.

A more critical bottleneck resides in routing congestion [15]. The wide buses carrying eigenvalue–eigenvector pairs (reaching 288 bits for $N = 8$) generate substantial fan-out and crossbar routing when switched between compare–exchange nodes, forcing place-and-route tools to adopt circuitous paths and significantly degrading F_{max} [16]. This physical-layer constraint proves more decisive than logical resource growth.

Recent research has pivoted toward alternative paths that curtail comparator counts. The comparison-free sorting scheme of Abdel-Hafeez and Gordon-Ross [17] encodes element ranks via counter arrays, collapsing time complexity to $O(N)$. Spatial complexity, however, inflates to $O(N^2)$. For an $N = 64$ system, the counter array alone consumes over 360,000 Look-Up Tables (LUTs), far exceeding the capacity of mid-range FPGAs. The multidimensional sorting algorithm Real-Time Hardware Sorter (RTHS) by Norollah et al. [18] alleviates spatial pressure through gridded data layout, yet its $O(N)$ latency growth remains prohibitive under real-time constraints. The Fast Lightweight Merge Sorter (FLiMS) proposed by Papaphilippou et al. [19] achieves breakthroughs in throughput. Its streaming architecture, however, resists adaptation to the batch-processing mode inherent to EVD. The survey by Jalilvand et al. [12] systematically charts this technological landscape, converging on a bottleneck that remains unresolved as follows: data movement overhead explodes when the objects being sorted expand from scalars to vectors.

2.2. Systolic Arrays and EVD Hardware Implementations

Kung’s foundational work in 1982 [7] codified the design philosophy of systolic arrays as follows: it regularized local interconnects and synchronized dataflow map algorithmic parallelism directly onto spatial hardware parallelism. Brent and Luk [8] applied this principle to Jacobi EVD, introducing the Round-Robin scheduling strategy that enables $n/2$ PEs to traverse all $n(n - 1)/2$ off-diagonal element pairs within $n - 1$ time steps. The elegance of this architecture resides in its complete dataflow determinism—interconnect topology is hardened at design time, obviating runtime routing decisions.

Subsequent research advanced along two axes. On the computational precision front, the introduction of Coordinate Rotation Digital Computer (CORDIC) algorithms [20,21] circumvented direct division and trigonometric evaluation by approximating rotation angles through iterative shift-accumulate operations. Mixed-precision strategies further balanced resource consumption against numerical error. On the architectural efficiency front, the parameterized pipeline design of Lightbody et al. [10] decoupled angle generation, rotation execution, and vector update into independent pipeline stages, establishing a template for subsequent Intellectual Property (IP) core development; Ding et al. [22] demonstrated a Fast Fourier Transform (FFT)-Singular Value Decomposition (SVD) joint accelerator on an Radio Frequency System-on-Chip (RFSoc) platform, validating the integration potential of systolic arrays within system-on-chip designs.

The core advantage of systolic arrays—static interconnects—paradoxically constitutes the obstacle to in-place sorting [6]. Embedding dynamic exchange logic would disrupt pipeline timing balance through conditional Multiplexer (MUX) chains, while long-wire delays and clock skew risks arising from cross-cell exchange signals would erode timing margins [23].

A common characteristic pervades these works as follows: the eigenvalue sorting problem is circumvented rather than confronted. The original Brent–Luk paper [8] left output ordering unaddressed. The subsequent literature either assumed that unordered outputs could be handled by higher-level software [10] or appended standalone sorting modules at the architectural periphery [22]. Eberlein’s analysis of one-sided Jacobi methods [24] and the rigorous convergence proofs by Usevich et al. [25] remained confined to the numerical stratum, never engaging hardware topology constraints. The convergence analyses of cyclic Jacobi strategies by Li et al. [26] and Begović et al. [27] illuminated the relationship between rotation ordering and convergence rate, yet neither extended these insights to the sorting scenario.

2.3. Limitations of Existing Approaches and Positioning of This Work

Examining these two research threads within the context of EVD applications reveals the following structural contradiction: hardware sorting networks are designed under the assumption of scalar sequences, whereas the true payload of EVD sorting comprises eigenvalue–eigenvector pairs. Consider an $N = 8$ system as follows: each eigenvalue is accompanied by a 256-bit eigenvector. Every exchange node in a bitonic sorter must therefore handle $32 + 256 = 288$ bits of data simultaneously. The resulting multiplexer overhead and routing congestion account for over 70% of the total sorting network resources. Najafi et al. [11] reduced comparison logic through unary encoding, yet the cost of wide-bus exchange remained untouched. This predicament scales quadratically with matrix dimension as follows: at $N = 32$, eigenvector width extends to 1024 bits, and even under the most resource-efficient RTHS scheme [18], the LUT consumption of the sorting module exceeds 150,000.

Within the systolic array itself, in-place sorting attempts founder on topological mismatch. The obstacle is not mathematical—the descending-order rotation strategy is theoretically sound, and the 0–1 principle [9] guarantees convergence for arbitrary inputs. The core barrier is the conflict between dynamic data permutation and static hardware routing as follows: the index swaps implicit in large-angle rotations disrupt the dataflow paths prescribed by Round-Robin scheduling. The existing literature neither identifies the topological nature of this conflict nor proposes an effective circumvention mechanism.

This work fills the aforementioned gap through the following paradigm shift: from “data movement” to “phase control.” The preceding analysis reveals that physical transport of wide buses constitutes the root cause of routing congestion and timing degradation.

This paper demonstrates that index inversions in Round-Robin scheduling adhere to a half-period pattern, which can be precisely compensated via sign-bit manipulation—the logical permutations required for sorting are thereby encoded as phase corrections rather than data movement, achieving, for the first time, zero-overhead in-place sorting within a systolic array.

3. Preliminaries

3.1. Principles of Jacobi Rotation

For an $n \times n$ real symmetric matrix $\mathbf{A}$, the Jacobi method diagonalizes it through a sequence of orthogonal similarity transformations [25,27]. Each transformation selects an index pair (p, q) and constructs a Givens rotation matrix $\mathbf{J}(p, q, \theta)$ such that the transformed matrix $\mathbf{A}' = \mathbf{J}^T \mathbf{A} \mathbf{J}$ annihilates the element at position (p, q) .

The rotation angle θ is determined by the following:

$$\tan(2\theta) = \frac{2a_{pq}}{a_{qq} - a_{pp}} \quad (1)$$

For the 2×2 submatrix, the post-rotation diagonal elements update as follows:

$$a'_{pp} = c^2 a_{pp} + s^2 a_{qq} - 2cs \cdot a_{pq} \quad (2)$$

$$a'_{qq} = s^2 a_{pp} + c^2 a_{qq} + 2cs \cdot a_{pq} \quad (3)$$

where $c = \cos \theta$ and $s = \sin \theta$.

3.2. Brent–Luk Systolic Array and Round-Robin Scheduling

The Brent–Luk architecture deploys $n/2$ PEs to execute Jacobi rotations in parallel [8]. Data scheduling adheres to the Round-Robin strategy, guaranteeing that all $n(n-1)/2$ off-diagonal element pairs are visited exactly once within $n-1$ time steps. For an eighth-order matrix, the scheduling sequence unfolds as follows:

$$\begin{bmatrix} (1,2) & (3,4) & (5,6) & (7,8) \\ (1,4) & (2,6) & (3,8) & (5,7) \\ (1,6) & (4,8) & (2,7) & (3,5) \\ (1,8) & (6,7) & (4,5) & (2,3) \\ (1,7) & (5,8) & (3,6) & (2,4) \\ (1,5) & (3,7) & (2,8) & (4,6) \\ (1,3) & (2,5) & (4,7) & (6,8) \end{bmatrix} \quad (4)$$

Each row corresponds to a single time step. Index pairs residing in the same row are processed concurrently.

4. Descending-Order Strategy and Problem Analysis

4.1. Descending-Order Mechanism via Arctan2

The conventional Jacobi algorithm computes rotation angles using the $\arctan$ function, whose range is restricted to $(-\pi/4, \pi/4]$ to ensure minimal rotation magnitude. The output ordering of eigenvalues, however, remains contingent upon the initial structure of the input matrix. This work achieves sorting control by reconfiguring the $\arctan2$ argument assignment as follows:

$$2\theta = \text{atan2}(-2a_{pq}, a_{pp} - a_{qq}) \quad (5)$$

Proposition 1 (Descending-Order Condition). *Under the above argument configuration, each rotation guarantees $a'_{pp} \geq a'_{qq}$.*

Proof. Let $X = a_{pp} - a_{qq}$, $Y = -2a_{pq}$, and $R = \sqrt{X^2 + Y^2}$. By the definition of $\arctan2$, $\cos(2\theta) = X/R$ and $\sin(2\theta) = Y/R$. Substituting into the Jacobi update formula yields the following:

$$a'_{pp} - a'_{qq} = \cos(2\theta)(a_{pp} - a_{qq}) - \sin(2\theta) \cdot 2a_{pq} = \frac{X^2 + Y^2}{R} = R > 0 \quad (6)$$

The larger eigenvalue is thus invariably placed at the smaller index. Upon convergence, a descending arrangement emerges naturally. $\square$

4.2. Theoretical Foundation and Verification of the Implicit Sorting Network

During the iterative process of Jacobi EVD combined with the Brent–Luk systolic array, all matrix elements simultaneously undergo numerical updates and positional transformations. This global perturbation appears entirely unrelated to conventional scalar sorting networks—the latter involve only pairwise comparisons and position exchanges, whereas the former constitutes complex orthogonal similarity transformations. Upon closer examination of the internal structure, the connection between the two becomes clearly discernible.

The evolutionary trajectory of off-diagonal elements is indeed difficult to predict; each Givens rotation alters their values, while Round-Robin scheduling circulates them throughout the entire array. Diagonal elements exhibit markedly different behavior. They remain confined to diagonal positions, undergoing deterministic position exchanges according to the Brent–Luk scheduling sequence. More critically, the essence of Jacobi iteration lies in progressively concentrating off-diagonal energy onto the diagonal—upon convergence, the diagonal elements become the target eigenvalues. This observation focuses the problem as follows: tracking only the relative magnitudes and positional evolution of diagonal elements suffices to capture the complete picture of sorting.

Consider an eighth-order matrix as an example. Extracting the diagonal elements $\{a_{11}, a_{22}, \dots, a_{88}\}$ to form an independent vector, and combining the pairwise-adjacent-element pairing rule in Round-Robin scheduling with the descending-order rotation strategy proposed in this work, an equivalent compare–exchange sequence can be constructed. The structure of this sequence is identical to the Round-Robin schedule presented in Section 3.2, as follows: each time step comprises four parallel comparisons, and seven time steps constitute one complete sweep.

The 0–1 principle from sorting network theory provides a rigorous guarantee for convergence [9] as follows: if a network correctly sorts all 2^N binary sequences, it is valid for arbitrary real-number sequences. For the $N = 8$ system, a complete test set encompassing all 256 zero-one patterns was constructed. Simulation results demonstrate that all test cases converge to strict descending order within three Round-Robin cycles (21 time steps). Under the idealized assumption that diagonal values remain fixed and undergo only positional exchanges, this suggests that three sweeps would suffice for sorting.

In practice, however, the dynamic coupling between numerical updates and positional exchanges complicates this model. Figure 1 illustrates the extraction process of the implicit sorting network and its verification under idealized conditions. As the convergence analysis in Section 7.4 demonstrates, during the first three to four rounds of Jacobi iteration, the numerical values of diagonal elements have not yet converged to the true eigenvalues but are in the process of gradual approximation. This implies that the magnitude relationship between two diagonal elements at a given moment may not coincide with their true ordering upon final convergence—the sorting network “compares” intermediate values that have

not yet stabilized, rather than the final answers. This dynamic coupling between numerical evolution and positional exchange cannot be exhaustively analyzed through analytical methods. Monte Carlo simulation is therefore employed for empirical verification.

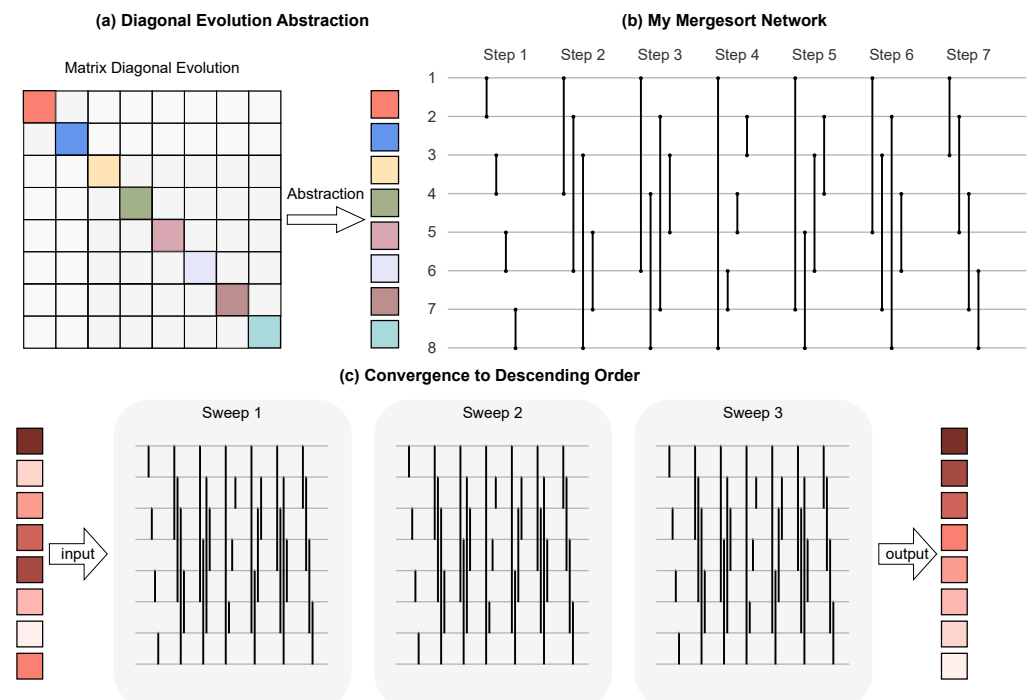


Figure 1. Extraction and verification of the implicit sorting network within the Jacobi process. (a) Diagonal element extraction: diagonal elements of the 8×8 matrix in the Brent–Luk systolic array are extracted as an independent vector, with different colors indicating the evolutionary trajectories of different eigenvalues. (b) Sorting network corresponding to a single sweep: vertical lines represent compare–exchange units, the network comprises 7 time steps, corresponding one-to-one with the adjacent-element pairing rules of Round-Robin scheduling. (c) Completeness verification over three sweeps: the left side shows an arbitrary initial permutation, which converges to strict descending order after 21 time steps (colors from dark to light represent values from large to small).

A random test of 10^6 trials was designed as follows: each trial generates a random real symmetric matrix, executes a specified number of Jacobi iteration rounds, and verifies whether the output eigenvalues satisfy strict descending order. Two auxiliary metrics were introduced to quantify sorting quality—Kendall’s τ coefficient measures the degree of inversion among element pairs ($\tau = 1$ indicates perfect descending order), and the relative norm error E_{norm} reflects the deviation from the true sorted order. Table 1 summarizes the statistical results.

Table 1. Sorting convergence statistics under one million Monte Carlo simulations.

	Complete Sorting Ratio	Mean Kendall’s τ	Mean Relative Norm Error
2 Jacobi Sweeps	61.10%	0.9661	8.79×10^{-2}
3 Jacobi Sweeps	98.22%	0.9987	3.05×10^{-3}
4 Jacobi Sweeps	100.00%	1.0000	5.91×10^{-6}

The data exhibit a clear convergence trend. At two sweeps, eigenvalues remain in an intensive energy redistribution phase, with only 60% of test cases achieving complete sorting. After three sweeps, this ratio surges to 98.22%; the minority of cases failing to achieve perfect sorting still exhibit τ coefficients as high as 0.9987—the actual deviation is negligible. Four sweeps achieve 100% strict descending order, with the τ coefficient consistently at 1.0.

Figure 2 further reveals the distributional details. The τ coefficient after three sweeps is highly concentrated near 1.0, with an extremely sparse long-tail region (panel a); the corresponding relative error distribution peaks at the 10^{-4} order of magnitude (panel b)—even when sorting is not entirely successful, the deviation remains within engineering-acceptable bounds. After four sweeps, the τ coefficient distribution degenerates into a single-point impulse at $\tau = 1.0$ (panel c), and the relative error peak further decreases to the 10^{-8} order of magnitude (panel d), approaching the precision limit of single-precision floating-point representation.

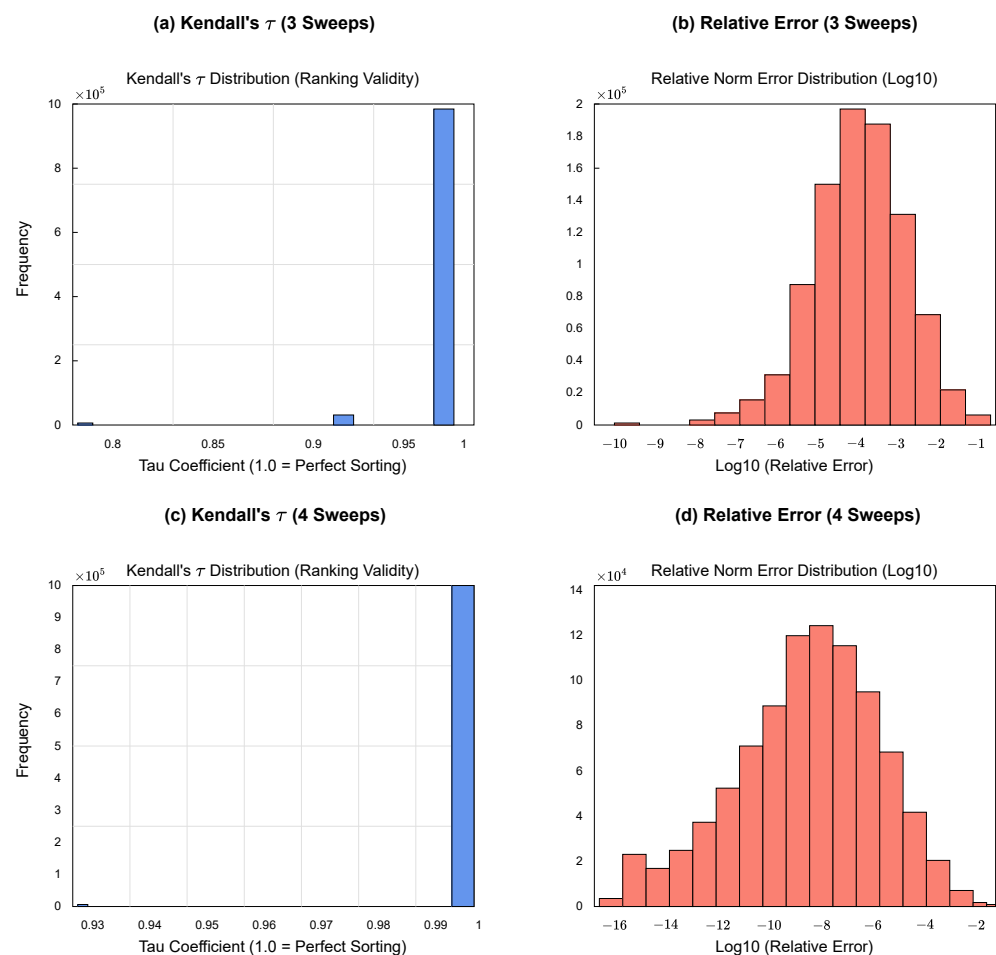


Figure 2. Statistical characteristics of implicit sorting under one million Monte Carlo simulations. (a) Distribution of Kendall's τ coefficient after 3 Jacobi sweeps. (b) Logarithmic distribution of relative norm error after 3 sweeps. (c) Distribution of Kendall's τ coefficient after 4 Jacobi sweeps. (d) Logarithmic distribution of relative norm error after 4 sweeps. $\tau = 1.0$ indicates perfect sorting; the peak of relative error decreases from the 10^{-4} order of magnitude at 3 sweeps to the 10^{-8} order at 4 sweeps.

The above results provide definitive guidance for hardware configuration. Three sweeps already satisfy the sorting requirements of the vast majority of real-time applications; the 1.78% failure rate can be absorbed by upper-layer software fault-tolerance mechanisms. Four sweeps completely eliminate sorting uncertainty, making them suitable for precision-sensitive scenarios. Given that additional iteration rounds yield no substantive improvement in sorting quality (having already reached 100%), exceeding four sweeps is not recommended in practical deployments.

4.3. Analysis of the Topological Mismatch Problem

The descending-order strategy is mathematically rigorous—yet its direct implementation within a Brent–Luk systolic array triggers severe topological mismatch.

When the rotation angle satisfies $|\theta| > \pi/4$, the Givens rotation matrix admits a factorization into a small-angle rotation composed with a permutation matrix as follows:

$$\mathbf{J}(\theta) = \hat{\mathbf{J}}(\phi) \cdot \mathbf{P}_{\text{swap}} \quad (7)$$

Here $\hat{\mathbf{J}}(\phi)$ denotes a small-angle rotation matrix ($|\phi| \leq \pi/4$), and $\mathbf{P}_{\text{swap}}$ is the permutation matrix that exchanges rows/columns p and q . A large-angle rotation, in other words, implicitly encodes a data exchange.

The interconnect topology of a systolic array, however, is static. Dataflow paths are uniquely determined by a preordained Round-Robin schedule. Denote this global permutation operator by Π . Under ideal conditions, the logical index evolves as $i^{(k+1)} = \Pi(i^{(k)})$. Once an implicit permutation π_{local} occurs inside a processing element, the actual evolution becomes $i_{\text{actual}}^{(k+1)} = \Pi(\pi_{\text{local}}(i^{(k)}))$. The operator π_{local} is data-driven—contingent upon the numerical values of matrix elements—whereas Π is hardwired at design time. The two admit no reconciliation. The theoretical underpinning of this mismatch traces back to convergence analyses of cyclic strategies in the Jacobi method [26,27].

The ramifications are immediate. Matrix $\mathbf{A}$ updates along the actual path $\Pi \circ \pi_{\text{local}}$; the eigenvector matrix $\mathbf{V}$ continues to update along the intended path Π . Divergence of these two trajectories corrupts the eigenvalue–eigenvector correspondence, rendering the algorithmic output invalid (as shown in Figure 3).

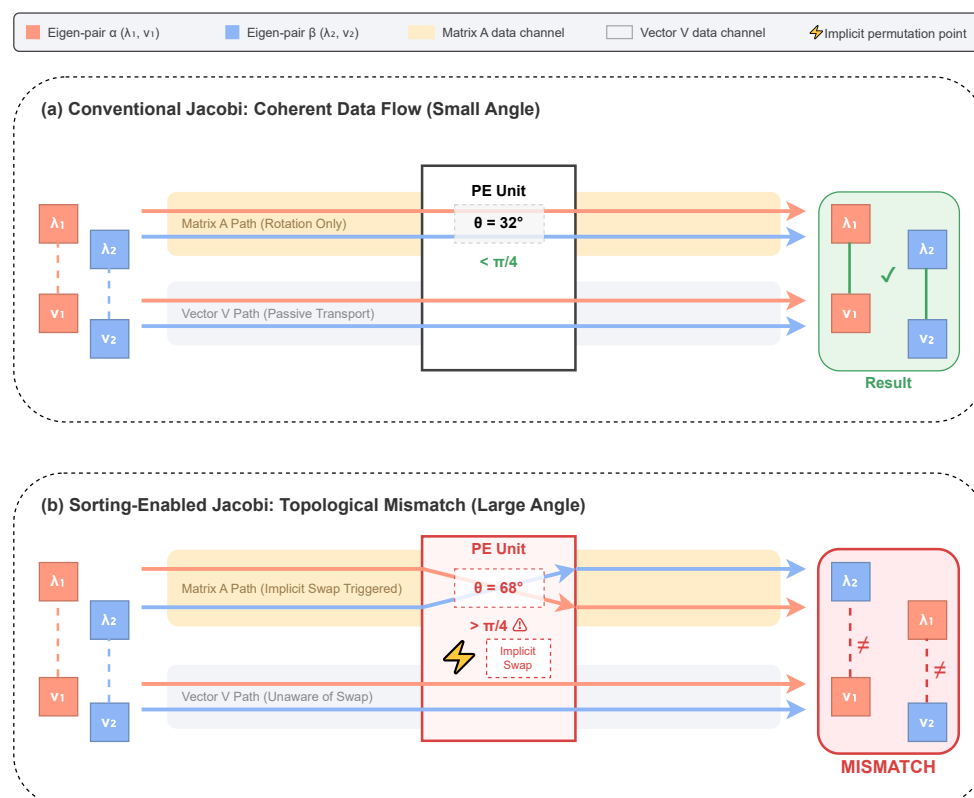


Figure 3. Topological mismatch phenomenon in a systolic array. (a) Under small-angle rotation, eigenvalue–eigenvector pairings remain intact. (b) Large-angle rotation induces an implicit permutation in the $\mathbf{A}$ path while eigenvector matrix $\mathbf{V}$ continues along its original trajectory, corrupting the correspondence.

5. Topological Phase Correction Algorithm

This section systematically derives the mechanism by which index reversal arises in systolic arrays and proposes a low-overhead correction strategy based on permutation matrices, fundamentally resolving the topological mismatch problem.

5.1. Mechanism of Index Reversal

In-depth analysis of the structural properties of Round-Robin scheduling sequences reveals a critical phenomenon: within specific time-step intervals, the index pairs held by processing elements undergo systematic reversal.

Proposition 2 (Half-Cycle Reversal Law). *For an n -th order Brent–Luk systolic array with standard initial ordering $(2k - 1, 2k)$, let $(p_k^{(t)}, q_k^{(t)})$ denote the index pair held by the k -th PE at time step t . Throughout this work, 1-based indexing is employed: time steps span $t \in \{1, 2, \dots, n - 1\}$, and PEs span $k \in \{1, 2, \dots, n/2\}$. Define the reverse-phase interval as $\frac{n}{2} < t < n$. Within this interval:*

1. *For all non-leading PEs ($k \geq 2$), the index pairs satisfy $p_k^{(t)} > q_k^{(t)}$;*
2. *The first PE P_1 exhibits invariance, always satisfying $p_1^{(t)} = 1 < q_1^{(t)}$.*

Taking an eighth-order matrix as an example, the complete Round-Robin scheduling sequence is:

$$\begin{bmatrix} (1, 2) & (3, 4) & (5, 6) & (7, 8) \\ (1, 4) & (2, 6) & (3, 8) & (5, 7) \\ (1, 6) & (4, 8) & (2, 7) & (3, 5) \\ (1, 8) & (6, 7) & (4, 5) & (2, 3) \\ (1, 7) & (8, 5) & (6, 3) & (4, 2) \\ (1, 5) & (7, 3) & (8, 2) & (6, 4) \\ (1, 3) & (5, 2) & (7, 4) & (8, 6) \end{bmatrix} \quad (8)$$

The bold entries mark processing elements experiencing index reversal, corresponding to time steps 5 through 7 (i.e., the interval $4 < t < 8$). This phenomenon is a topological inevitability of the serpentine permutation path, originating from the “fixed vertex, rotating remainder” construction rule inherent to Round-Robin scheduling.

Matrix symmetry ($a_{pq} = a_{qp}$) renders index reversal inconsequential for numerical correctness. The geometric meaning of the rotation matrix, however, is altered—downstream dataflow is consequently affected. This is precisely the root cause demanding correction.

5.2. Phase Classification of Processing Elements

Based on the magnitude relationship of index pairs, we partition PEs into the following two principal categories: diagonal units and off-diagonal units. The phase distribution is illustrated in Figure 4.

(1) Diagonal Units

PEs residing on the main diagonal (P_{kk}) bifurcate into two types according to index ordering as follows:

- *Type I (Normal Phase):* Satisfying $p_k < q_k$. Only the first processing element P_1 belongs to this category; its internal data exhibit a normal-order layout as follows:

$$\mathbf{M}^{(I)} = \begin{bmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{bmatrix}, \quad p < q \quad (9)$$

- *Type II (Reverse Phase)*: Satisfying $p_k > q_k$. All subsequent diagonal units P_k ($k \geq 2$) fall into this category. Internal data exhibit a reverse-order layout—main and anti-diagonal positions are swapped, as follows:

$$\mathbf{M}^{(II)} = \begin{bmatrix} a_{qq} & a_{qp} \\ a_{pq} & a_{pp} \end{bmatrix}, \quad p > q \quad (10)$$

(2) Off-Diagonal Units

PEs at off-diagonal positions ($P_{ij}, i \neq j$) derive their data layout from the phase combination of their row diagonal unit P_{ii} and column diagonal unit P_{jj} as follows:

- *Type 1 Off-Diagonal Units (Heterogeneous)*: Row and column diagonal units possess differing phases (one normal, one reverse). The element layout manifests a “single-sided flip” as follows:

$$\mathbf{M}^{(1)} = \begin{bmatrix} a_{p_i p_j} & a_{p_i q_j} \\ a_{q_i p_j} & a_{q_i q_j} \end{bmatrix} \xrightarrow{\text{column flip}} \begin{bmatrix} a_{p_i q_j} & a_{p_i p_j} \\ a_{q_i q_j} & a_{q_i p_j} \end{bmatrix} \quad (11)$$

- *Type 2 Off-Diagonal Units (Homogeneous)*: Both row and column diagonal units are Type II (reverse phase). The element layout undergoes a “double flip” as follows:

$$\mathbf{M}^{(2)} = \begin{bmatrix} a_{p_i p_j} & a_{p_i q_j} \\ a_{q_i p_j} & a_{q_i q_j} \end{bmatrix} \xrightarrow{\text{row-column flip}} \begin{bmatrix} a_{q_i q_j} & a_{q_i p_j} \\ a_{p_i q_j} & a_{p_i p_j} \end{bmatrix} \quad (12)$$

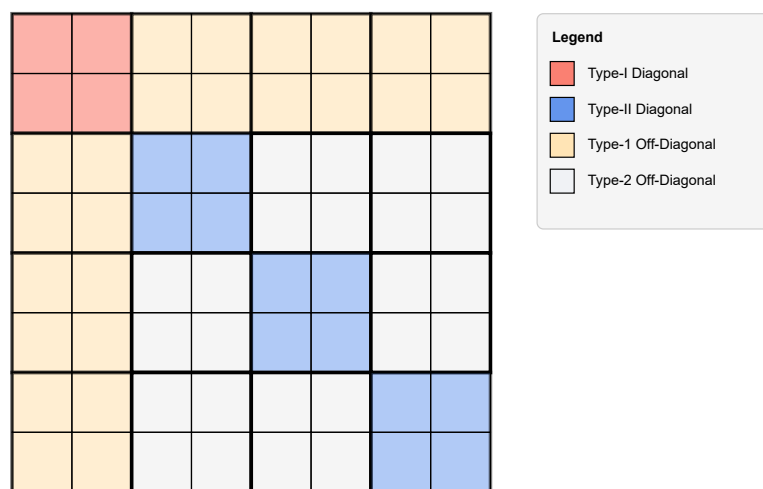


Figure 4. Phase distribution diagram of the systolic array. Type I diagonal units (P_{11} , red region) and Type II diagonal units (P_{kk} , blue region) possess distinct phase characteristics. These two core unit categories, through their respective row/column broadcast mechanisms, govern the phase correction modes for remaining off-diagonal units (apricot and gray regions).

5.3. Topological Correction Derivation Based on Standard Process

Dataflow and rotation factor propagation in systolic arrays are intricately intertwined. Direct analysis of correction logic easily becomes mired in index tracking. To establish a clear derivation framework, this section introduces the standard process as a reference frame—temporarily setting aside the serpentine transfer structure of systolic arrays, we adopt only its core concept of 2×2 sub-block parallel computation, extracting four non-conflicting off-diagonal element pairs directly from the matrix for simultaneous rotation.

The mathematical formulation of the standard process is as follows. Select the coordinates (i, j) of the off-diagonal element to be eliminated and construct the corresponding Givens rotation as follows:

$$\begin{pmatrix} \alpha'_{pq} & \beta'_{pq} \\ \gamma'_{pq} & \delta'_{pq} \end{pmatrix} = \begin{pmatrix} c_p & -s_p \\ s_p & c_p \end{pmatrix} \begin{pmatrix} \alpha_{pq} & \beta_{pq} \\ \gamma_{pq} & \delta_{pq} \end{pmatrix} \begin{pmatrix} c_q & s_q \\ -s_q & c_q \end{pmatrix} \quad (13)$$

Eigenvector synchronous update follows $\mathbf{V}' = \mathbf{V}\mathbf{J}$. Since the update matrix $\mathbf{J}$ affects only two columns of $\mathbf{V}$, if the four actively selected index pairs are mutually non-overlapping, four rotations can operate on the eigenvector matrix in full parallelism. This update scheme represents the classical form of the Jacobi method [8,25,28]. We term the aforementioned joint update of eigenvalues and eigenvectors the standard process (std), defining its rotation factors as (c_s, s_s) —these factors can be directly applied to eigenvector updates without any correction.

We now shift perspective to the actual systolic array. The original rotation factors computed via $\arctan2$ at diagonal units are denoted (c_o, s_o) . The phase classification in Section 5.2 indicates that Type I diagonal units ($p < q$) exhibit data layouts fully consistent with the standard process. Hence, $(c_s, s_s) = (c_o, s_o)$, with factors passed through directly. Type II diagonal units ($p > q$) require careful analysis as follows: their internal data exhibits a reverse-order layout with main and anti-diagonal positions swapped. Substituting into the angle computation formula

$$\theta = \frac{1}{2} \arctan\left(\frac{2a_{pq}}{a_{pp} - a_{qq}}\right) \quad (14)$$

reveals that the denominator sign is inverted. This sign change, after mapping through $\arctan2$, is equivalent to taking the complementary angle of the computed angle θ_o relative to the standard angle θ_s at $\pm 90^\circ$, i.e., $\theta_o + \theta_s = \pm \pi/2$. From trigonometric identities, we obtain the following:

$$c_s = |s_o|, \quad s_s = \text{sgn}(s_o)c_o \quad (15)$$

At this point, the correction of diagonal units toward the standard process is complete. The corrected (c_s, s_s) can be directly employed for eigenvector updates, while the eigenvector matrix $\mathbf{V}$ resides in an independent register array accessed via dynamic indexing; its physical layout does not participate in systolic flow, thus requiring no element transfer as with matrix $\mathbf{A}$.

We next derive the correction rules for off-diagonal units. Introduce the permutation matrix $\mathbf{E} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, where right-multiplication implements column interchange, left-multiplication implements row interchange, and self-inverse property $\mathbf{E}^2 = \mathbf{I}$ holds.

Consider Type 1 off-diagonal units as an example, where column indices undergo reversal, yielding the relationship $\mathbf{A}_{curr} = \mathbf{A}_{std}\mathbf{E}$ between current and standard layouts. Let the standard process output be $\mathbf{B}_{std} = \mathbf{G}_1\mathbf{A}_{std}\mathbf{G}_2$; we seek corrected rotation matrices $\mathbf{G}'_1, \mathbf{G}'_2$ such that the output $\mathbf{B}_{curr}$ preserves the same topological structure, i.e., $\mathbf{B}_{curr} = \mathbf{B}_{std}\mathbf{E}$. Right-multiplying both sides of the standard update formula by $\mathbf{E}$ and inserting $\mathbf{E}\mathbf{E} = \mathbf{I}$ to the right of $\mathbf{A}_{std}$ via the following self-inverse property:

$$\mathbf{B}_{std}\mathbf{E} = \mathbf{G}_1\mathbf{A}_{std}(\mathbf{E}\mathbf{E})\mathbf{G}_2\mathbf{E} \implies \mathbf{B}_{curr} = \mathbf{G}_1(\mathbf{A}_{std}\mathbf{E})(\mathbf{E}\mathbf{G}_2\mathbf{E}) \quad (16)$$

Comparison yields the following: the left-multiplication matrix $\mathbf{G}'_1 = \mathbf{G}_1$ remains unchanged. The right-multiplication matrix requires similarity transformation $\mathbf{G}'_2 = \mathbf{E}\mathbf{G}_2\mathbf{E}$.

For Type 2 off-diagonal units, both row and column indices undergo reversal, giving $\mathbf{A}_{curr} = \mathbf{E}\mathbf{A}_{std}\mathbf{E}$. Similarly, left- and right-multiplying both sides of the standard update formula by $\mathbf{E}$ and inserting identity transformations in between, we derive that both left and right rotation matrices require the EGE transformation.

Expanding the explicit form of EGE as follows:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \quad (17)$$

Comparing the result with the standard rotation matrix $\begin{bmatrix} c_m & -s_m \\ s_m & c_m \end{bmatrix}$ immediately yields the correction rule: $c_m = c_s, s_m = -s_s$.

Synthesizing the above analysis, the correction logic exhibits a clear hierarchical structure, as follows: if rotation factors originate from a Type II diagonal unit direction, the sine term requires negation; factors from Type I diagonal unit directions pass through directly. The physical interpretation of this rule is equally intuitive—the EGE transformation essentially applies a “mirror conjugation” to the rotation matrix, precisely compensating for the geometric deviation introduced by the reverse-order layout.

5.4. Unified Phase Correction Logic

The preceding section established mappings from (c_o, s_o) to (c_s, s_s) for diagonal units and from (c_s, s_s) to (c_m, s_m) for off-diagonal units, respectively. In practical hardware, the Angle Generation Unit (AGU) directly outputs (c_o, s_o) , necessitating single-step generation of both corrected factors (c_m, s_m) for matrix updates and standard factors (c_s, s_s) for eigenvector updates. Composing these two-layer mappings yields the unified truth table shown in Table 2.

Taking $\theta_o = 30^\circ$, i.e., $c_o = 0.866, s_o = 0.500$ as an example, the corresponding truth table is as follows:

Table 2. Unified truth table for phase correction.

Phase State	Condition	Matrix Rotation Factors (c_m, s_m)	Vector Rotation Factors (c_s, s_s)
Type I (Normal Phase)	Always applicable	(0.866, 0.5)	(0.866, 0.5)
Type II (Reverse Phase)	$s_o = 0.5 \geq 0$	(0.5, −0.866)	(0.5, 0.866)
Type II (Reverse Phase)	$s_o < 0$ (N/A)	-	-

The simplicity of this truth table is no coincidence. It is underpinned by deep algebraic structure. We examine the computational kernel of Type II correction as follows: computing $|s_o|$ reduces to conditional negation controlled by the sign bit, $-c_o$ involves fixed negation, and the difference between (c_m, s_m) and (c_s, s_s) manifests solely in the sign bit of the s component. In other words, the entire correction logic compresses to three XOR operations and two 2-to-1 multiplexers.

This minimalist form originates from the coupling of two mathematical properties. First is the orthogonality constraint of Givens rotations, outlined as follows: all four elements of the rotation matrix are completely determined by a single angle θ , with sine and cosine values rigidly coupled. Any correction must preserve this constraint. Second is the self-inverse property of permutation matrices, outlined as follows: $\mathbf{E}^2 = \mathbf{I}$ implies that two consecutive identical permutations cancel each other, endowing correction operations with “reversibility”—if a direction requires correction, the inverse operation in that direction is precisely reapplying the same correction. It is precisely this “symmetry dividend” that reduces the seemingly complex topological correction to bit-level operations.

5.5. Consistency Analysis of Eigenvector Updates

The two parallel sets of output factors in Table 2— (c_m, s_m) and (c_s, s_s) —do not constitute redundant design, but they rather originate from the fundamentally distinct storage and update mechanisms of matrix **A** and eigenvector matrix **V** within the systolic array.

The data of matrix **A** flow along the serpentine path of the systolic array, with their 2×2 sub-blocks assigned to different PEs at different time steps. Analysis in Section 5.2 demonstrates that sub-block layouts within the reverse-phase interval deviate from the predetermined order of the standard process. The corrected factors (c_m, s_m) serve precisely to compensate for this deviation as follows: while executing the Givens rotation, they implicitly apply permutation **E** to local sub-blocks, enabling output data to continue flowing correctly along the predetermined path. In this sense, (c_m, s_m) constitutes a composite operator, fusing geometric rotation with topological correction.

The situation for eigenvector matrix **V** differs entirely. As described in Section 6.3, **V** resides in an independent fully partitioned register array, with designated column vectors accessed on demand through a dynamic indexing network. This “data stationary, pointer mobile” architecture implies that the physical storage layout of **V** remains fixed at all times, without participating in the systolic array’s element transfers. Since no topological offset arises from dataflow, updates to **V** naturally need only reflect pure geometric rotation angles—precisely the meaning of standard factors (c_s, s_s) .

Erroneously applying (c_m, s_m) to update **V** would be equivalent to imposing an unnecessary permutation **E** on its columns, disrupting the eigenvalue–eigenvector correspondence. The correctness of the standard process [8,25,28] guarantees that **V** updated with (c_s, s_s) consistently satisfies $\mathbf{AV} = \mathbf{V}\mathbf{\Lambda}$.

5.6. Hardware Implementation and Cost Analysis

Based on the mapping rules in Table 2, the hardware implementation of the Phase Correction Unit (PCU) exhibits minimalist structural characteristics. During actual iterations, the original matrix undergoes systolic flow according to the standard Brent–Luk architecture, with each PE employing (c_m, s_m) to execute matrix updates. The eigenvector matrix remains stationary in place, receiving (c_s, s_s) in parallel to complete column vector rotations, without requiring element transfers as in traditional architectures. The sole modification involves computing angles via $\arctan$ and executing corrections per Table 2 prior to rotation factor broadcasting.

Taking the computation of (c_m, s_m) for the Type II phase as an example, the core operations decompose into four steps as follows:

1. **Sign Detection:** Extract the most significant bit (sign bit) of s_0 to determine whether $s_0 \geq 0$ or $s_0 < 0$.
2. **Conditional Negation :** If $s_0 < 0$, perform bitwise inversion on s_0 and add one to obtain $|s_0|$.
3. **Fixed Negation:** Execute negation on c_0 to obtain $-c_0$.
4. **Cross Assignment:** $c_m \leftarrow |s_0|$, $s_m \leftarrow -c_0$.

The generation of (c_s, s_s) shares the first two steps with the above procedure, differing only in sign combination at the fourth step. Type I phase completely bypasses correction logic, directly passing through original factors via multiplexers.

The required hardware resources are summarized in Table 3.

Under fixed-point representation, negation operations are implemented via XOR gate arrays, with conditional negation accomplished through sign-bit-controlled adders. The entire correction circuit comprises pure combinational logic, consuming neither Digital Signal Processing (DSP) nor Block Random Access Memory (BRAM) resources, with an equivalent area of approximately 10–20 LUTs. This module is inserted between the AGU

and Rotation Execution Unit (REU). Its combinational path delay is substantially less than floating-point multiply–accumulate operations, thus not constituting a critical path bottleneck, leaving the maximum system operating frequency unchanged.

Table 3. Hardware resource requirements for the Phase Correction Unit.

Component	Quantity	Functional Description
Sign Bit Extraction	1	Read most significant bit of s_o
Conditional Negator	1	Compute $\ s_o\ $ (negate when $s_o < 0$)
Fixed Negator	1	Compute $-c_o$
2:1 MUX	2	Type I/II factor selection (bypass or correction)

Remark 1 (Virtual Rewiring Perspective). *The phase correction operation ($s \rightarrow -s$) proposed in this work can be viewed, in physical essence, as a form of “virtual rewiring” applied to the systolic array. Without altering the underlying fixed interconnect routing, algebraic sign flipping dynamically adjusts the topological attributes of local dataflow, adapting it to the logical permutations that arise during sorting. This mechanism effectively resolves the conflict between dynamic dataflow and static hardware architecture.*

6. Hardware Implementation

6.1. Overall Architecture

Figure 5 depicts the proposed EVD processor architecture [22]. The system comprises the following three core modules: a pre-processing unit, a systolic array core, and a post-processing unit.

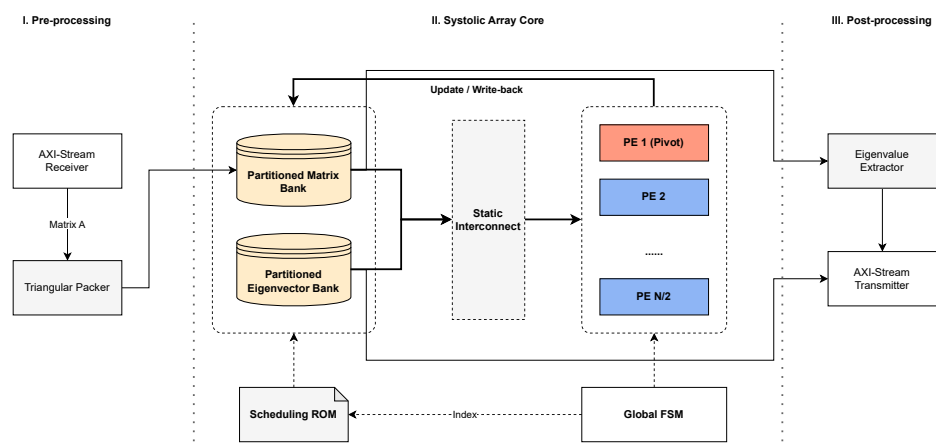


Figure 5. Proposed EVD processor architecture [22]. The system integrates a pre-processing unit for storage compression, a systolic array core for parallel Jacobi rotations, and a post-processing unit for result extraction.

The pre-processing module transforms the input $n \times n$ real symmetric matrix into a one-dimensional upper-triangular storage format. Matrix symmetry permits the storage of merely $n(n + 1)/2$ elements, yielding an approximate 50% reduction in on-chip memory resources. The systolic array core instantiates $n/2$ fully pipelined PEs that execute Jacobi rotations and phase corrections in parallel. The post-processing module extracts diagonal elements as eigenvalue outputs and, when required, exports the eigenvector matrix.

Data routing employs a static network to sustain high throughput. The Round-Robin scheduling sequence is precomputed and embedded within read-only memory (ROM) as a lookup table (Pulse_mapping). At runtime, the control logic queries this table directly to

generate data-exchange addresses—complex dynamic routing computations are thereby circumvented, and control-path latency is minimized.

6.2. PE Design

Figure 6 illustrates the microarchitecture of an individual PE, which cascades the following three pipeline stages: the Angle Generation Unit (AGU), the Phase Correction Unit (PCU), and the Rotation Execution Unit (REU).

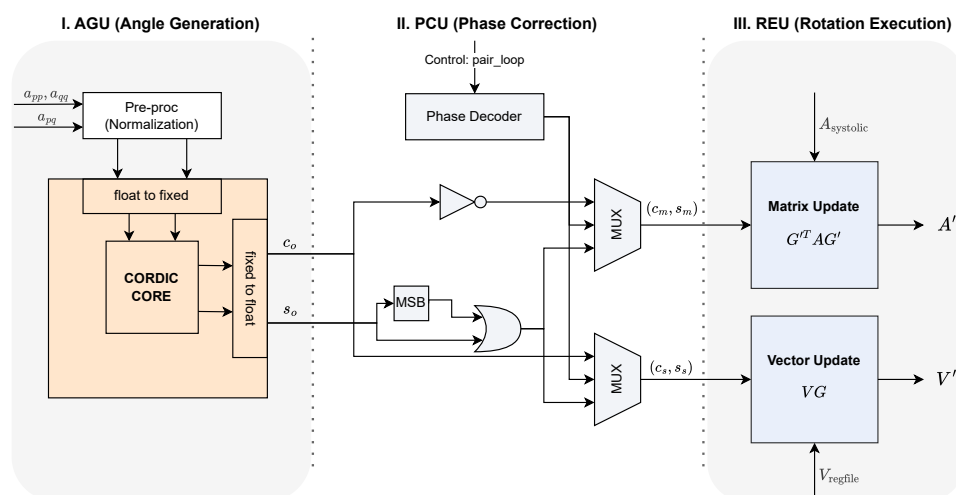


Figure 6. Microarchitecture of an individual PE, showing the overall pipeline cascading AGU, PCU, and REU, along with the internal logic of the functional units.

(1) *Angle Generation Unit (AGU)*: A mixed-precision strategy reconciles numerical accuracy with resource economy. Input data undergo normalization and conversion to a 24-bit fixed-point format before entering the CORDIC module [20,21], which computes the original rotation factors (c_o, s_o). This approach curtails both the latency and the resource footprint of transcendental-function evaluation relative to a pure floating-point implementation.

(2) *Phase Correction Unit (PCU)*: Situated between the AGU and REU, the PCU inspects the global scheduling index *pair_loop* to ascertain the current PE’s phase state—Type I or Type II—and executes the mapping logic codified in Table 2. Leveraging the broadcast strategy of Section 5.3, the correction process reduces to bit-level manipulations: the sign bit of s_o steers a multiplexer that emits the corrected factors (c_m, s_m) alongside the standard geometric factors (c_s, s_s). The module consists entirely of combinational logic and introduces no additional pipeline latency.

(3) *Rotation Execution Unit (REU)*: The REU incorporates a dual-path update structure [10] as follows: the *matrix path* receives (c_m, s_m) and executes the transformation $\mathbf{G}^T \mathbf{A} \mathbf{G}'$; the *vector path* receives (c_s, s_s) and executes $\mathbf{V} \mathbf{G}$. The computational kernel houses four single-precision floating-point multipliers and two adders, all fully pipelined to achieve single-cycle throughput.

6.3. Eigenvector Storage and Parallel Update

To eliminate the data-movement bottleneck inherent in eigenvector updates and to satisfy the ultra-low-latency demands of edge computing, this work introduces a Decoupled Parallel Update Architecture. The architecture exploits the control-flow characteristics of the phase correction algorithm to achieve physical separation between computation and storage.

(1) High-Bandwidth Fully Partitioned Register File

For the $N = 8$ edge-side real-time processing scenarios targeted in this work—5G/6G beamforming being a representative example—storage bandwidth constitutes the principal throughput constraint. Conventional BRAM-based schemes, limited to dual-port read/write access, inevitably introduce serialization delays. We therefore adopt a fully partitioned register file strategy. This design incurs a modest increase in logic resources yet delivers fully parallel access bandwidth, enabling $N/2$ rotation units to acquire all operands within a single clock cycle. The eigenvector matrix $\mathbf{V}$ consequently updates in complete synchrony with the systolic computation of matrix $\mathbf{A}$, entirely eliminating pipeline bubbles attributable to memory access.

(2) Zero-Latency Dynamic Interconnect

The fully parallel storage architecture transforms physical data movement under Round-Robin scheduling into lightweight logical index mapping. We designed a LUT-based dynamic interconnect network that, according to the scheduling indices (p, q) stored in ROM, routes column vectors $\mathbf{v}_p$ and $\mathbf{v}_q$ directly from the register file to the computational units.

The REU incorporates an independent vector update path that receives the standard geometric factors (c_s, s_s) from the PCU. The system instantiates $N/2$ vector rotation units in parallel, executing the following:

$$\begin{bmatrix} \mathbf{v}'_p & \mathbf{v}'_q \end{bmatrix} = \begin{bmatrix} \mathbf{v}_p & \mathbf{v}_q \end{bmatrix} \begin{bmatrix} c_s & -s_s \\ s_s & c_s \end{bmatrix} \quad (18)$$

This “data-stationary, pointer-flowing” strategy realizes zero hardware-level latency for eigenvector updates, fully decoupled from the systolic flow of matrix $\mathbf{A}$.

(3) Scalability and Storage Agnosticism

The topological phase correction strategy proposed in this work is inherently storage-medium agnostic. External sorting networks that rely on high-bandwidth data exchange would incur read/write conflicts when implemented with BRAM. The present scheme, by contrast, internalizes sorting as phase control over rotation angles—sign-bit logic alone—without any additional physical data movement. For large-scale matrix applications ($N \geq 64$) requiring BRAM, the algorithm migrates seamlessly and introduces no extra read/write overhead, exhibiting excellent hardware scalability.

6.4. Storage and Dataflow Optimization

To match the $n/2$ -way parallel computation demands of the systolic array, we employ a cyclic partitioning storage strategy. Using the High-Level Synthesis (HLS) directive `#pragma HLS ARRAY_PARTITION`, upper-triangular matrix data are fully unrolled and mapped onto independent register banks. This guarantees that all PEs can simultaneously obtain their four required operands (a_{pp} , a_{qq} , a_{pq} , and overlapping elements) within each clock cycle, eliminating the memory-access bottleneck.

Figure 7 illustrates the complete dataflow mapping from logical layout to parallel execution. In the left-side 8×8 upper-triangular matrix, both diagonal and off-diagonal regions are partitioned at the granularity of 2×2 sub-blocks; following cyclic partitioning, identically colored elements are allocated to independent register banks (center), providing full-bandwidth support for $n/2$ -way parallel access. In the right-side architecture, four AGU groups complete angle computation concurrently within a single clock cycle; their outputs, after parallel correction by PCUs, drive two independent paths for matrix update and eigenvector update, respectively. Units of the same type (e.g., four AGUs) execute in parallel, while heterogeneous units (AGU $\rightarrow$ PCU $\rightarrow$ REU) cascade in a pipelined fashion.

This fully partitioned register strategy circumvents the bandwidth bottleneck inherent in BRAM dual-port access, enabling $n/2$ -way computational units to simultaneously acquire all required operands within a single cycle, thereby eliminating computational stalls induced by memory access conflicts.

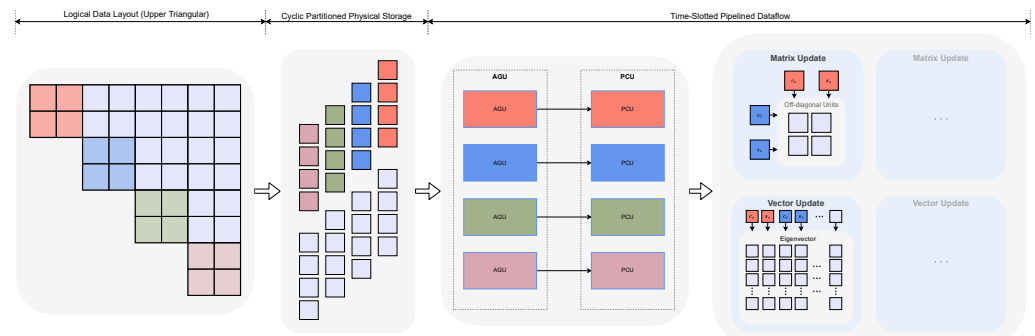


Figure 7. Unified dataflow and phase distribution in the Brent–Luk systolic array. **(Left)** Cyclic partitioning of the upper-triangular matrix; **(Center)** Mapping to independent register banks; **(Right)** Parallel execution pipeline showing AGU, PCU, and REU stages.

The entire iterative process adopts a four-stage pipeline design: angle computation, phase correction, matrix rotation, and eigenvector update overlap temporally. For an eighth-order matrix, a single complete Round-Robin cycle requires only 7 time steps; four cycles (sweeps) totaling 28 time steps suffice to achieve both high-precision convergence and strict descending-order sorting.

7. Experimental Results

7.1. Experimental Setup

Hardware implementation and verification were conducted within the Xilinx Vitis HLS 2024.1 development environment, targeting the Xilinx Zynq UltraScale+ RFSoc device xczu47dr-fvvg1517-2-i.

The test dataset comprises randomly generated eighth-order real symmetric matrices encoded in standard IEEE 754 single-precision floating-point format (32-bit). Jacobi iteration is fixed at four complete Round-Robin cycles—28 time steps in total. This configuration serves a dual purpose, as follows: it guarantees convergence to engineering-grade accuracy (error $< 10^{-4}$) while permitting the algorithm sufficient iterations to complete the “energy transport” inherent to the sorting process. C/RTL co-simulation validated hardware logic correctness and yielded precise resource consumption and latency metrics.

The synthesis target clock period was set to 5.0 ns (200 MHz). Vitis 2024.1 reported positive worst-case negative slack, with the achievable $F_{\max}$ reaching 250 MHz. Under the four-sweep configuration, a single complete EVD operation consumes 10,193 cycles. At a 200 MHz operating frequency, this yields a throughput of approximately 19.6k matrices/s for 8×8 matrices (single-matrix latency: 51.0 μ s).

7.2. Resource Consumption and Efficiency Analysis

Table 4 contrasts the resource footprint of the proposed sorting-enabled Jacobi architecture against a traditional non-sorting baseline. A canonical external sorting module—an $N = 8$ bitonic sorting network—is included as a reference benchmark to crystallize the engineering value of our approach. Figure 8 visualizes the overhead differential across key resources (LUT and Flip-Flop (FF)).

Table 4. Resource consumption and performance comparison for an eighth-order EVD processor.

Core Metric	Traditional Arch. (No Sorting)	Proposed Arch. (With Sorting)	Proposed Overhead	External Sorting (Bitonic Sort)	Proposed Advantage
LUT (Logic)	46,582	46,953	+371	4120	91% Reduction
FF (Registers)	31,280	31,550	+270	3840	92% Reduction
DSP (Comp.)	136	136	0	0	—
BRAM (Storage)	36	36	0	0	—
Latency (Cycles)	10,193	10,193	0	47	Eliminates Add. Latency

Notes : 1. External sorting module resource estimates are based on a complete bitonic sorting network incorporating eigenvalue comparison and synchronous transfer of a 256-bit eigenvector bus [11,13]. 2. The $N = 8$ bitonic network comprises 24 compare-exchange nodes (6 stages, 4 nodes per stage). Each node processes 288-bit-wide data (32-bit eigenvalue + 256-bit eigenvector), incurring approximately 172 LUTs (dual-path MUX + comparator) and 160 FFs (pipeline registers) per node—totaling $24 \times 172 \approx 4120$ LUTs and $24 \times 160 = 3840$ FFs. 3. Latency amounts to approximately 8 cycles per stage (compare, select, and register), yielding 47 cycles across 6 stages (the first stage requires no additional input registration).

Figure 8 reveals that the bar heights for our architecture and the baseline (no sorting) are virtually indistinguishable—the LUT/FF increment remains below 1%. On the latency front, elimination of the external sorter renders the hardware processing latency of both architectures identical under the four-sweep high-precision setting. Sorting capability is thus acquired at zero time cost and zero area cost.

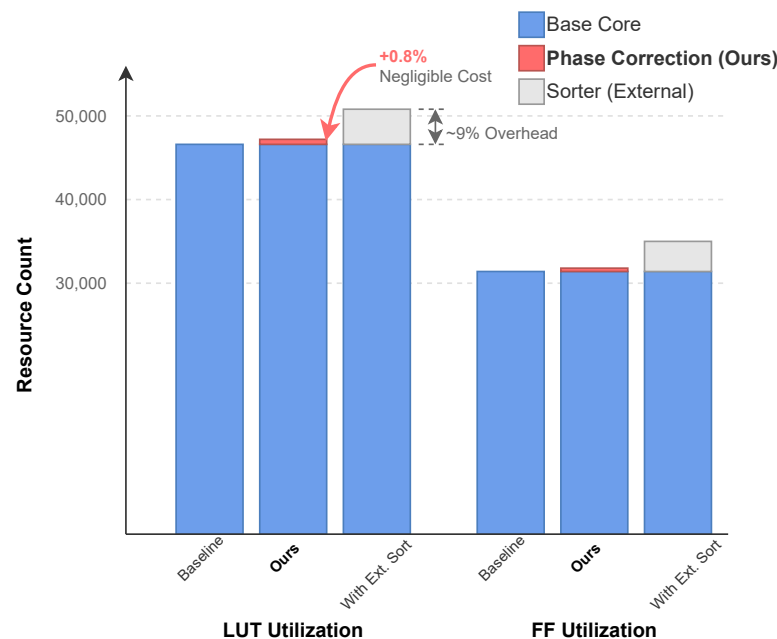


Figure 8. Resource overhead comparison for an $N = 8$ EVD processor. Compared with the non-sorting baseline, the proposed architecture incurs negligible logic overhead (+0.80% LUTs, +0.86% FFs) and consumes no additional DSP slices or BRAM blocks; sorting is achieved entirely through control-flow modification rather than arithmetic expansion. In contrast, a standard bitonic sorter requires approximately 4120 LUTs to handle the physical relocation of 256-bit eigenvector buses—the proposed virtual rewiring mechanism eliminates this bottleneck, yielding a 91% reduction in sorting-related logic.

(1) Minimal Intrinsic Overhead

Quantitative inspection confirms what the bar chart suggests, outlined as follows: our architecture introduces merely 371 additional LUTs (+0.80%) and 270 additional FFs (+0.86%). This marginal increment stems from sign-bit detection and conditional logic

within the Phase Correction Unit (PCU). Critically, neither DSP slices nor BRAM tiles are consumed. The implication is clear—modifying the rotation-angle computation strategy endows the hardware with sorting capability at near-zero cost.

The combinational-path delay of phase correction logic—comprising elementary XOR gates and 2-to-1 multiplexers—falls well below that of floating-point multiply-accumulate (MAC) units. The critical path therefore remains entirely dominated by floating-point arithmetic, and the maximum operating frequency (F_{max}) suffers no degradation upon insertion of the correction circuit.

(2) Decisive Advantage over External Sorting

Sorting in EVD hardware transcends simple eigenvalue comparison [12,17]. The genuine challenge lies in synchronously relocating the associated eigenvector matrix. The significantly higher “External Sorting” bar in Figure 8 illustrates this burden as follows:

- *Wide-bus exchange overhead:* Each eigenvalue in an $N = 8$ system is paired with an eigenvector spanning eight single-precision floats—256 bits in aggregate. Every exchange node within the sorting network must therefore shuttle this full width. Constructing the requisite switching fabric consumes approximately 4120 LUTs.
- *Superiority of the proposed approach:* The “compute-as-sort” strategy embeds sorting within the rotation computation itself. A sign-bit flip on the rotation factor logically permutes data without physically moving the 256-bit eigenvector payload. Compared with external bitonic sorting, this mechanism reduces logic resource consumption by approximately 91%.

7.3. Latency Performance Analysis

The proposed architecture achieves zero additional pipeline latency per iteration. Sorting logic resides entirely within the parallel computation path. The hardware time cost of a single sweep is therefore identical to that of a conventional non-sorting architecture.

Under the unified four-sweep test configuration, total processing latency for the proposed design matches the baseline exactly—both complete in 10,193 cycles. An external sorting solution, by contrast, would not only incur the substantial area overhead documented in Section 7.2 but would inevitably append additional pipeline stages to execute physical data relocation. For a 256-bit-wide eigenvector bus, this penalty typically spans several tens of cycles. The proposed scheme thus elevates system functional integration while preserving equivalent computational throughput.

7.4. Convergence and Accuracy Validation

To comprehensively evaluate the numerical performance of the proposed architecture, we conducted systematic comparisons between hardware simulation results, MATLAB (2024b) double-precision reference solutions, and a conventional non-sorting Jacobi algorithm across 10^6 randomly generated 8×8 real symmetric matrices.

The hardware-generated eigenvalues invariably satisfy the strict monotonic ordering $d_1 \geq d_2 \geq \dots \geq d_8$. This 100% success rate confirms the precise steering capability of the phase correction mechanism over the dataflow and demonstrates that sorting functionality is achieved entirely without external comparator networks.

Convergence accuracy is characterized by two complementary metrics. Normalized off-diagonal energy E_{off} is defined as the ratio of the sum of squared off-diagonal elements to the squared Frobenius norm of the matrix, directly reflecting the progress of Jacobi iteration toward diagonalization. Maximum relative error $E_{rel} = \max_i |(\lambda_i^{HW} - \lambda_i^{Ref}) / \lambda_i^{Ref}|$ quantifies the deviation of hardware-computed eigenvalues from double-precision ground truth; this metric decouples from input data magnitude, objectively reflecting the number of

significant digits preserved by the algorithm. Table 5 documents the complete convergence trajectories of both architectures across varying iteration counts.

Table 5. Convergence performance comparison ($N = 10^6$).

Sweeps	Eoff (Proposed)	Eoff (Baseline)	Erel (Proposed)	Erel (Baseline)
1	5.33×10^{-1}	3.94×10^{-1}	$1.25 \times 10^{+1}$	$7.78 \times 10^{+0}$
2	2.12×10^{-1}	7.80×10^{-2}	$5.03 \times 10^{+0}$	9.88×10^{-1}
3	2.26×10^{-2}	2.88×10^{-3}	2.04×10^{-1}	3.38×10^{-3}
4	2.59×10^{-4}	3.85×10^{-6}	1.63×10^{-4}	2.95×10^{-7}
5	1.64×10^{-7}	4.17×10^{-11}	6.66×10^{-9}	1.20×10^{-14}

Figure 9 presents a comparison of the two convergence curves on logarithmic axes. During Sweeps 1–3, the proposed architecture exhibits significantly higher E_{off} and E_{rel} than the conventional scheme—this is not an algorithmic deficiency but rather an inherent cost of the sorting mechanism. The system undergoes intensive “global energy transport,” forcibly migrating larger eigenvalues from the lower-right region of the matrix to the upper-left. Such cross-matrix data permutations impose transient perturbations on the diagonalization process.

The impact of these perturbations diminishes rapidly after Sweep 4. Once sorting stabilizes, the convergence rate of the proposed architecture accelerates markedly as follows: at Sweep 4, E_{off} plummets to the 10^{-4} order of magnitude, and E_{rel} reaches 1.63×10^{-4} —sufficient for the vast majority of engineering applications. Sweep 5 data indicate that continued iteration yields further gains as follows: E_{rel} drops to the 10^{-9} order, approaching the precision limit of single-precision floating-point representation.

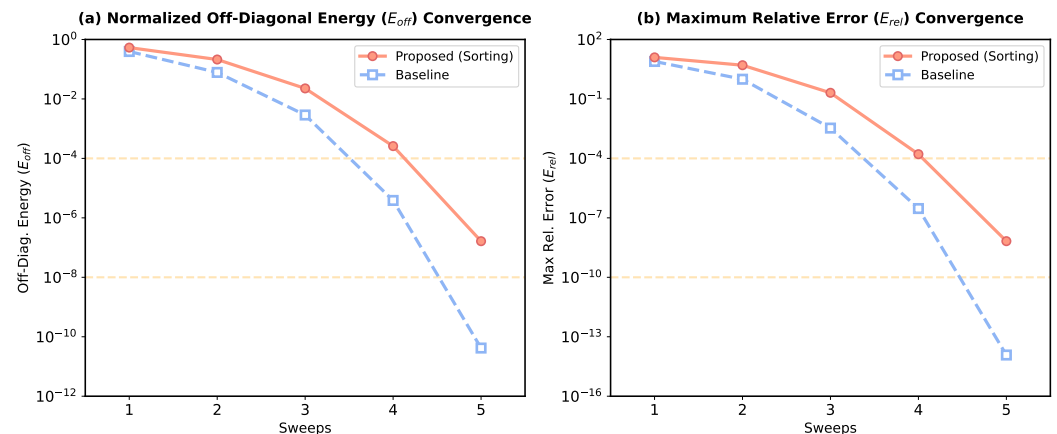


Figure 9. Convergence curve comparison between the proposed sorting-enabled architecture and the conventional non-sorting baseline across 5 sweeps. (a) Normalized Off-Diagonal Energy (E_{off}) convergence; (b) Maximum Relative Error (E_{rel}) convergence.

Beyond numerical convergence of eigenvalues, the impact of the phase correction strategy on eigenvalue–eigenvector pairing relationships warrants verification. Table 6 presents statistical results for the following two complementary metrics: decomposition residual and orthogonality error.

Table 6. Eigenvalue–eigenvector coupling relationship validation.

Validation Metric	Mathematical Definition	3 Sweeps (Mean)	4 Sweeps (Mean)
Decomposition Residual	$\frac{\ A\mathbf{V} - \mathbf{V}\Lambda\ _F}{\ \mathbf{A}\ _F}$	2.25×10^{-2}	2.52×10^{-4}
Orthogonality Error	$\ \mathbf{V}^T \mathbf{V} - \mathbf{I}\ _F$	1.59×10^{-15}	1.84×10^{-15}

The decomposition residual directly verifies the defining equation $\mathbf{AV} = \mathbf{V}\mathbf{\Lambda}$ as follows: at three sweeps, the residual of 2.25×10^{-2} is commensurate with eigenvalue error itself (Table 5), indicating that pairing deviations primarily originate from insufficient iteration convergence. At four sweeps, the residual plunges to the 10^{-4} order, confirming that pairing relationships are entirely correct once high-precision convergence is attained. Orthogonality error validates the correction strategy from the numerical stability perspective—this error remains locked at 10^{-15} (double-precision machine epsilon floor) and is decoupled from iteration count, confirming that the sign-bit operations in Table 2 precisely preserve the orthogonal structure of the Givens matrix.

Synthesizing Tables 5 and 6, the proposed scheme meets expectations along both numerical precision and structural correctness dimensions. The convergence characteristics documented above furnish clear configuration guidelines for engineering deployment as follows: for latency-sensitive applications (e.g., real-time precoding in millimeter-wave beamforming), four sweeps delivers engineering-grade precision at the 10^{-4} level; precision-sensitive applications (e.g., high-resolution Direction of Arrival (DOA) estimation) may adopt five sweeps, trading seven additional time steps for 10^{-9} -level high-precision output.

7.5. Comprehensive Comparison of Hardware Resources and Efficiency

To validate the superiority of the proposed method, we selected the following three representative hardware sorting architectures as benchmarks: the classical bitonic sorting network [11], the RTHS [18], and the comparison-free One-hot encoding sorting [17].

All comparative data are grounded in an $N = 8$ EVD application scenario. The critical engineering constraint is as follows: sorting involves not merely eigenvalue comparison (32-bit Float) but must also synchronously relocate the corresponding eigenvector ($8 \times 32 = 256$ -bit). This “wide-bus payload” dramatically amplifies the hardware overhead of conventional sorting networks. Resource estimates are derived from synthesis characteristics of the Xilinx UltraScale+ architecture. Table 7 presents the hardware overhead and complexity comparison.

Table 7. Hardware overhead and complexity comparison of different sorting schemes at $N = 8$ scale.

Scheme	Core Mechanism	Time Comp.	Space Comp.	LUT	FF	Latency	Norm. ADP
Bitonic [11,13]	compare-exchange net	$O(\log^2 N)$	$O(N \log^2 N)$	4120	3840	47	260.4
RTHS [18]	Multidim. grid sort	$O(N)$	$O(N)$	2850	2100	32	270.3
One-hot [17]	Counter array	$O(N)$	$O(N^2)$	5600	4100	8	144.6
This Work	Topo. phase corr.	$O(1)$	$O(N)$	371	270	0	1.0

Notes: 1. Area-Delay Product (ADP) is computed as $\text{LUT} \times (\text{Latency} + 1)$ and normalized to the proposed scheme. 2. Time complexity denotes the growth order of additional latency introduced by sorting. Space complexity denotes the growth order of hardware resources consumed by sorting logic. 3. Time complexity $O(1)$ indicates that sorting introduces no additional latency. Space complexity $O(N)$ arises from constant logic added to each PE, with total overhead scaling linearly with the number of PEs ($N/2$)—still substantially lower than the $O(N^2)$ or $O(N \log^2 N)$ of conventional schemes. 4. Bitonic data are based on a fully pipelined architecture. Latency includes $\frac{1}{2} \log_2 N (\log_2 N + 1) = 6$ levels of comparison logic and register overhead. 5. RTHS data are proportionally scaled from the $N = 16$ measured results reported in [18].

(1) Complexity and Scalability Analysis

Table 7 exposes fundamental differences in scalability across schemes. The bitonic network incurs $O(N \log^2 N)$ space complexity. Scaling the matrix dimension from $N = 8$ to $N = 64$ inflates the required compare-exchange nodes from 24 to 672—a 28-fold resource explosion. The $O(N^2)$ space complexity of the One-hot scheme proves

even more prohibitive, as follows: at $N = 64$, 4096 counter units are demanded, driving LUT consumption beyond 360,000—far exceeding the capacity of mid-range FPGAs.

In contrast, the proposed scheme exhibits $O(1)$ time complexity (zero additional latency) and $O(N)$ space complexity. The former derives from sorting logic being fully fused into the Jacobi iteration pipeline, introducing no wait cycles whatsoever. The latter arises because the phase correction circuitry within each PE requires only sign-bit detection and conditional negation logic, with total overhead scaling linearly with the $N/2$ PEs. This $O(N)$ resource growth retains an order-of-magnitude advantage over the $O(N^2 \log^2 N)$ scaling of bitonic sorting.

(2) Resource Inflation Induced by “Wide-Bus” Payloads

Table 7 reveals that while the bitonic network possesses structural regularity, each switching node must process a total data width of 288 bits ($32 + 256$). Constructing 24 such wide-bus switching nodes in an FPGA implementation consumes over 4000 LUTs, predominantly allocated to extensive MUX logic. The RTHS scheme reduces comparator count through algorithmic refinement, yet interconnect complexity when handling wide vectors still incurs approximately 2850 LUTs.

(3) Latency–Pipeline Trade-offs

The One-hot scheme achieves $O(N)$ time complexity (8 cycles), yet its $O(N^2)$ space complexity drives LUT consumption to 5600—unacceptable for edge-side devices. The bitonic network constrains latency to 47 cycles through deep pipelining, but this still constitutes additional processing overhead.

(4) Summary of Advantages of the Proposed Scheme

The phase correction strategy proposed in this work fundamentally circumvents the “data relocation” problem. By modifying only the sign bit of the rotation factor (a 1-bit operation), we logically accomplish permutation of a 256-bit-wide vector. This explains why our scheme achieves equivalent functionality with merely 371 LUTs—a 91% reduction compared to bitonic sorting. More critically, this process is fully fused into the Jacobi iteration pipeline, achieving zero additional latency (time $O(1)$) and linear resource growth (space $O(N)$)—the ADP metric remains nearly two orders of magnitude superior to conventional schemes.

7.6. Scalability Analysis

The resource growth characteristics of the proposed scheme differ fundamentally from those of conventional bitonic sorting networks. The core overhead of phase correction logic originates solely from sign-bit operations within each diagonal PE (approximately 50 LUT/PE) and broadcast routing. The total increment approximates $O(N)$. A bitonic sorter, by contrast, requires $\frac{N}{2} \cdot \frac{\log_2 N (\log_2 N + 1)}{2}$ compare–exchange nodes, each of which must handle an $(N + 1) \times 32$ -bit-wide bus (including synchronous eigenvector relocation), inducing $O(N^2 \log^2 N)$ resource growth. Table 8 compares LUT consumption of the two schemes across varying scales as follows: at $N = 16$, the proposed scheme requires 782 LUTs whereas bitonic sorting demands approximately 26,400 LUTs—a 97.0% reduction. At $N = 32$, the proposed scheme consumes 1657 LUTs while bitonic sorting balloons to approximately 153,600 LUTs (already exceeding the capacity of most mid-range FPGAs)—a 98.9% reduction. This advantage arises from encoding sorting as sign-bit correction, thereby completely circumventing the physical relocation of wide-bus vectors.

Table 8. Resource increment comparison of sorting schemes across matrix scales.

Matrix Dim. N	Proposed (LUT)	Bitonic (LUT)	Savings
8	371	4120	91.0%
16	782	26,400	97.0%
32	1657	153,600	98.9%

8. Discussion

8.1. Routing Constitutes the True Bottleneck

FPGA resource evaluation frequently focuses on LUT consumption—a perspective that proves misleading in the context of sorting networks. Modern FPGAs possess relatively abundant logic resources. The true constraint on system performance often resides in routing resources [15,16].

The predicament of bitonic sorting networks originates precisely from this constraint. Consider $N = 8$; each compare–exchange node must process 288-bit-wide data (32-bit eigenvalue + 256-bit eigenvector). When the comparison result dictates an exchange, these 288 bits must switch between two paths. The MUX tree implementing this functionality not only consumes logic resources but, more critically, its high fan-out characteristics force the routing tool to adopt circuitous paths. The dataflow directions of 24 exchange nodes are dynamically determined by runtime results, forming a global interconnection pattern that readily induces routing congestion and degrades F_{max} [16].

The proposed scheme exhibits fundamentally different interconnection characteristics. All phase correction operations—sign-bit detection, conditional negation, multiplexer selection—occur within a single PE, with corrected rotation factors requiring transmission only to the REU within the same PE. This nearest-neighbor interconnection pattern aligns seamlessly with the design philosophy of systolic arrays as follows: dataflows along fixed paths, control signals are generated and consumed locally, obviating the need for long-distance transmission across regions. Since the correction logic comprises pure combinational circuitry with extremely short paths, its delay is entirely masked by floating-point operations in the REU and does not constitute a critical path.

A design principle thereby emerges as follows: on FPGAs, avoiding data movement is more fundamental than optimizing movement efficiency. The cost of bitonic networks derives not from the comparison logic itself, but from the wide-bus data exchanges triggered by comparison results. The proposed scheme redefines “sorting” as “phase control,” fundamentally circumventing physical data relocation.

8.2. Scalability Advantages

As matrix dimension N increases, the gap between the two schemes widens dramatically. The data in Table 8 clearly illustrate this trend as follows: as N scales from 8 to 32, the proposed scheme’s LUT consumption grows from 371 to 1657, whereas bitonic sorting inflates from 4120 to 153,600—the latter already approaching the capacity ceiling of mid-range FPGAs.

The root cause of this disparity lies in the fundamental difference in growth order. The node count of bitonic networks scales as $O(N \log^2 N)$, with each node processing a $(N + 1) \times 32$ -bit-wide data bus, resulting in superlinear resource expansion. In the proposed scheme, the correction logic within each PE—sign-bit extraction, conditional negation, multiplexer selection—is entirely independent of N . Whether the matrix is 8×8 or 64×64 , the circuit scale of a single PCU remains constant. Total overhead grows only linearly with the number of PEs ($N/2$), and the growth coefficient is minimal.

More critically, consider the contrast in pipeline depth. Each doubling of scale in a bitonic network necessitates additional pipeline stages, with latency accumulating accordingly. The sorting logic of the proposed scheme is fully fused into the existing computational flow, introducing no additional pipeline stages—this implies that regardless of how N varies, the latency penalty for sorting functionality remains consistently zero.

9. Conclusions

This paper reveals the root cause of in-place sorting failure in systolic arrays as follows: an irreconcilable conflict exists between the implicit data permutation induced by large-angle Givens rotations and the fixed routing prescribed by Round-Robin scheduling—we define this as topological mismatch. The key to resolving this problem lies in recognizing that index reversal is not random perturbation but adheres to a precise half-cycle inversion pattern. This deterministic structure enables the complex permutation correction to collapse into single-bit sign operations, equivalent to accomplishing “virtual rewiring” without altering physical interconnects.

Fusing sorting functionality into the Jacobi iteration process yields fundamental hardware simplification. Conventional approaches cascade an independent sorting network after the EVD core—comprising comparator arrays, wide-bus multiplexers, and their cross-region interconnects—the proposed scheme eliminates these modules entirely. FPGAs no longer require dedicated hardware resources allocated for sorting functionality; the sorting logic that formerly occupied thousands of LUTs is compressed into sign-bit detection circuits distributed across individual processing elements, whose resource consumption is nearly negligible. More critically, data movement is circumvented as follows: the physical exchange of the eigenvector matrix (256 bits wide for $N = 8$) is replaced by phase correction, thereby eradicating the routing congestion induced by high fan-out routing. Sorting and computation execute in complete overlap, introducing no additional pipeline stages—the hardware latency per sweep remains identical to that of a non-sorting architecture.

FPGA experiments corroborate the above analysis. The PCU introduces merely 0.8% additional LUT resources while eliminating 91% of the logic overhead of external sorting networks. The cost of this “compute-as-sort” strategy manifests in the convergence trajectory as follows: global energy redistribution imposes transient perturbations during early iterations, with E_{rel} converging to the 10^{-4} order of magnitude after four sweeps—satisfying the requirements of the vast majority of engineering applications.

The results of this work demonstrate that the topological constraints of systolic arrays are not obstacles to functional extension but rather structural resources that can be exploited at the algorithmic level. The core concept of the virtual rewiring mechanism—leveraging the determinism of scheduling patterns to predict and compensate for dataflow deviations—holds promise for generalization to other Givens rotation-based matrix computation hardware, including SVD and QR decomposition.

Author Contributions: Conceptualization, C.T. and Y.C.; Methodology, C.T.; Validation, C.T. and Y.L.; Investigation, C.T., B.L. and Y.L.; Data curation, C.T. and B.L.; Writing—Original draft, C.T. and Y.L.; Writing—Review and editing, C.T., Y.C., Y.L. and M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: All data generated or analyzed during this study are included in this published article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

EVD	Eigenvalue Decomposition
PE	Processing Element
FPGA	Field-Programmable Gate Array
MIMO	Multiple Input Multiple Output
VLSI	Very Large Scale Integration
CORDIC	Coordinate Rotation Digital Computer
AGU	Angle Generation Unit
PCU	Phase Correction Unit
REU	Rotation Execution Unit
ROM	Read-Only Memory
HLS	High-Level Synthesis
LUT	Look-Up Table
FF	Flip-Flop
DSP	Digital Signal Processing
BRAM	Block Random Access Memory
ADP	Area-Delay Product
RTHS	Real-Time Hardware Sorter
GPU	Graphics Processing Unit
IP	Intellectual Property
SVD	Singular Value Decomposition
FFT	Fast Fourier Transform
RFSoc	Radio Frequency System-on-Chip
DOA	Direction of Arrival
FLiMS	Fast Lightweight Merge Sorter

References

1. Dai, L.; Tan, J.; Poor, H.V. Delay-phase precoding for wideband THz massive MIMO. *IEEE Trans. Commun.* **2021**, *70*, 7271–7286. [\[CrossRef\]](#)
2. Ning, B.; Tian, Z.; Mei, W.; Chen, Z.; Han, C.; Li, S.; Yuan, J.; Zhang, R. Beamforming technologies for ultra-massive MIMO in terahertz communications. *IEEE Open J. Commun. Soc.* **2023**, *4*, 614–658. [\[CrossRef\]](#)
3. Gao, Z.; Hu, C.; Dai, L.; Wang, Z. Channel estimation for millimeter-wave massive MIMO with hybrid precoding over frequency-selective fading channels. *IEEE Commun. Lett.* **2016**, *20*, 1259–1262. [\[CrossRef\]](#)
4. Merkofer, J.P.; Revach, G.; Shlezinger, N.; Routtenberg, T.; van Sloun, R.J.G. DA-MUSIC: Data-driven DoA estimation via deep augmented MUSIC algorithm. *IEEE Trans. Signal Process.* **2022**, *70*, 2178–2190. [\[CrossRef\]](#)
5. Rokhlin, V.; Szlam, A.; Tygert, M. A randomized algorithm for principal component analysis. *SIAM J. Matrix Anal. Appl.* **2009**, *31*, 1100–1124. [\[CrossRef\]](#)
6. Heath, R.W.; Gonzalez-Prelcic, N.; Rangan, S.; Roh, W.; Sayeed, A.M. An overview of signal processing techniques for millimeter wave MIMO systems. *IEEE J. Sel. Top. Signal Process.* **2016**, *10*, 436–453. [\[CrossRef\]](#)
7. Kung, H.T. Why systolic architectures? *Computer* **1982**, *15*, 37–46. [\[CrossRef\]](#)
8. Brent, R.P.; Luk, F.T. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. *SIAM J. Sci. Stat. Comput.* **1985**, *6*, 69–84. [\[CrossRef\]](#)
9. Knuth, D.E. *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2nd ed.; Addison-Wesley: Reading, MA, USA, 1998. [\[CrossRef\]](#)
10. Lightbody, G.; Woods, R.; Walke, R. Design of a parameterizable silicon intellectual property core for QR-based RLS filtering. *IEEE Trans. VLSI Syst.* **2003**, *11*, 659–678. [\[CrossRef\]](#)
11. Najafi, M.H.; Lilja, D.J.; Riedel, M.D.; Bazargan, K. Low-cost sorting network circuits using unary processing. *IEEE Trans. VLSI Syst.* **2018**, *26*, 1471–1480. [\[CrossRef\]](#)
12. Jalilvand, A.H.; Banitaba, F.S.; Estiri, S.N.; Aygun, S.; Najafi, M.H. Sorting it out in hardware: A state-of-the-art survey. *ACM Trans. Des. Autom. Electron. Syst.* **2025**, *30*, 1–31. [\[CrossRef\]](#)
13. Peters, H.; Han, T.; Kopp, O. Bitonic sort on the GPU. In Proceedings of the High Performance Graphics 2009, New Orleans, LA, USA, 1–3 August 2009; pp. 108–117.

14. Mueller, R.; Teubner, J.; Alonso, G. Sorting networks on FPGAs. *VLSI J.* **2012**, *21*, 1–23. [[CrossRef](#)]
15. Zhao, J.; Liang, T.; Sinha, S.; Zhang, W. Machine Learning Based Routing Congestion Prediction in FPGA High-Level Synthesis. *arXiv* **2019**, arXiv:1905.03852. [[CrossRef](#)]
16. Betz, V.; Rose, J.; Marquardt, A. *Architecture and CAD for Deep-Submicron FPGAs*; Kluwer Academic Publishers: Boston, MA, USA, 1999.
17. Abdel-Hafeez, S.; Gordon-Ross, A. An efficient $O(n)$ comparison-free sorting algorithm. *IEEE Trans. VLSI Syst.* **2017**, *25*, 1930–1942. [[CrossRef](#)]
18. Norollah, A.; Derafshi, D.; Beitollahi, H.; Fazeli, M. RTHS: A low-cost high-performance real-time hardware sorter, using a multidimensional sorting algorithm. *IEEE Trans. VLSI Syst.* **2019**, *27*, 1601–1613. [[CrossRef](#)]
19. Papaphilippou, P.; Brooks, C.; Luk, W. FLiMS: Fast lightweight merge sorter. In *2018 International Conference on Field-Programmable Technology (FPT)*; IEEE: Maui, HI, USA, 2018; pp. 78–85. [[CrossRef](#)]
20. Pilato, L.; Fanucci, L.; Saponara, S. Real-time and high-accuracy arctangent computation using CORDIC and fast magnitude estimation. *Electronics* **2017**, *6*, 22. [[CrossRef](#)]
21. Nawandar, N.K.; Satpute, V.R. A study and comparison of COordinate Rotation DIgital Computer (CORDIC) architectures. *arXiv* **2022**, arXiv:2211.04053.
22. Ding, H.; Kang, C.C.; Xi, S.Y.; Liu, Z.; Zhang, X.; Ding, Y. FPGA-Optimized Hardware Accelerator for Fast Fourier Transform and Singular Value Decomposition in AI. In *2024 International Conference on Computing Innovation, Intelligence, Technologies and Education (CIITE)*; IEEE: New York, NY, USA, 2024; pp. 1–5. [[CrossRef](#)]
23. Peltekis, C.; Filippas, D.; Dimitrakopoulos, G.; Nicopoulos, C.; Pnevmatikatos, D. ArrayFlex: A Systolic Array Architecture with Configurable Transparent Pipelining. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Antwerp, Belgium, 17–19 April 2023; pp. 1–6.
24. Eberlein, P.J. On one-sided Jacobi methods for parallel computation. *SIAM J. Alg. Disc. Meth.* **1987**, *8*, 790–796. [[CrossRef](#)]
25. Usevich, K.; Li, J.; Comon, P. Approximate matrix and tensor diagonalization by unitary transformations: Convergence of Jacobi-type algorithms. *SIAM J. Matrix Anal. Appl.* **2020**, *41*, 1728–1750. [[CrossRef](#)]
26. Li, J.; Usevich, K.; Comon, P. Globally convergent Jacobi-type algorithms for simultaneous orthogonal symmetric tensor diagonalization. *SIAM J. Matrix Anal. Appl.* **2018**, *39*, 1–22. [[CrossRef](#)]
27. Begović, E.; Hari, V. Jacobi method for symmetric 4×4 matrices converges for every cyclic pivot strategy. *Numer. Algorithms* **2018**, *78*, 701–720. [[CrossRef](#)]
28. Yan, D.; Wang, W.X.; Zhang, X.W. High-Performance Matrix Eigenvalue Decomposition Using the Parallel Jacobi Algorithm on FPGA. *Circuits Syst. Signal Process.* **2023**, *42*, 1573–1592. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.