

Article

High-Performance Multithreaded Acceleration of High-Order Discrete Racah Polynomials on Multi-Core Architectures

Raafat Salih Muhammad ¹, Hayder S. Radeaf ¹, Hala J. Mohammed ¹, Ahlam Hanoon Al-sudani ¹,
Basheera M. Mahmmod ¹, Mazin S. AlHakeem ², Muntadher Alsabah ³ and Sadiq H. Abdulhussain ^{1,*}

¹ Department of Computer Engineering, University of Baghdad, Al-Jadriya, Baghdad 10071, Iraq; raafat.muhammad@coeng.uobaghdad.edu.iq (R.S.M.); haydersaadi@coeng.uobaghdad.edu.iq (H.S.R.); hala.jassim@coeng.uobaghdad.edu.iq (H.J.M.); assis.prof.a.hanoon@coeng.uobaghdad.edu.iq (A.H.A.-s.); basheera.m@coeng.uobaghdad.edu.iq (B.M.M.)

² Department of Computer Engineering, Al-Farabi University, Baghdad 10022, Iraq; mazin.alhakeem@alfarabiuc.edu.iq

³ Medical Technical College, Al-Farahidi University, Baghdad 10071, Iraq; muntadher.m@uofarahidi.edu.iq

* Correspondence: sadiqhabeeb@coeng.uobaghdad.edu.iq

Abstract

Discrete Racah polynomials (DRPs) are considered essential for several applications such as approximation theory, quantum physics, and digital signal processing. However, the computation of high-order DRPs presents significant numerical and computational challenges. Even though the recent advancements in algorithms have improved numerical stability, the inherently sequential nature of these algorithms continues to be a bottleneck for large-scale implementations. This paper proposes a new multithreading-based algorithm for computing the DRP coefficients. The proposed algorithm is designed to accelerate the computation of high-order DRPs by considering the advantages of the independence of different tasks and the ability to process multiple coefficients simultaneously. The algorithm introduces distinct multithreading strategies for both zero-valued and non-zero-valued parameters, which include the zero-parameter balanced algorithm (ZPBA), the zero-parameter unbalanced algorithm (ZPUA), and the valued-parameter algorithm (VPA). The experimental results show that the proposed multithreading algorithms noticeably reduce computation time when compared to the state-of-the-art baseline algorithm. Notably, both ZPBA and VPA exhibit exceptional scalability across modern multicore architectures, delivering speedup factors exceeding $10\times$ for large polynomial sizes. This work effectively bridges the gap between mathematical theory and high-performance computing, making the generation of large-scale DRP matrices computationally viable for advanced engineering applications.

Keywords: discrete Racah polynomials; orthogonal polynomials; orthogonal moments; parallel processing; multi-threading



Academic Editor: Binlin Zhang

Received: 23 June 2026

Revised: 22 July 2026

Accepted: 24 July 2026

Published: 26 July 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.

1. Introduction

The rapid escalation of modern signal processing and computational complexity has been linked to the development and application of high-order discrete orthogonal polynomials. Discrete Racah polynomials (DRPs) hold a unique position among discrete orthogonal polynomials due to their placement at the top of the Askey scheme's discrete branch. Consequently, many other polynomial families are derived from DRPs as limiting cases. Originally introduced by Giulio Racah in the context of angular momentum theory

in quantum mechanics, DRPs are deeply connected to $6j$ -symbols and representation theory [1–3].

The mathematical properties of DRPs have allowed for advanced applications in quantum physics [3], approximation theory and spectral methods [4], digital signal processing [5], image processing and reconstruction [6–9], 3D object estimation [10], watermarking and data protection [11–14], and pattern classification [15–17].

However, computing DRPs of higher orders entails substantial numerical and computational challenges. The definition of DRPs involves generalized hypergeometric functions as well as multiple recurrence relations. This leads to high computational complexity and numerical instability as the polynomial order (degree) increases [4]. In high-dimensional and large-scale problems such as spectral transforms, feature extraction, and orthogonal moment computation; the sequential generation of higher-order DRPs becomes a computational bottleneck [18].

The development of computer systems utilizing central processing units (CPUs), especially multicore processors and shared-memory parallel architectures, has made multithreading a practical strategy for accelerating numerically intensive algorithms [19–21]. In particular, thread-level parallelism (parallel computing paradigms) is implemented via frameworks such as OpenMP and POSIX threads. This implementation will enable the distribution of independent computational tasks across processor cores. This strategy will surely reduce the execution time for compute-bound operations as well as enhance the overall efficiency of numerical computations [22–24]. Polynomial-based transforms and recurrence formulas contain the potential for parallel computation. These can be utilized in the computation process of the polynomial coefficients, recurrence evaluation, and the construction of transformation matrices [25].

Although DRPs are important in advanced mathematical modeling and several applications, limited focus has been paid to the high-performance implementation of efficiently handling higher-order computations. Most of the existing works have focused on the development of implementations that emphasize theoretical formulations rather than computational scalability. This has resulted in a gap between mathematical theory and efficient high-performance implementation.

In this paper, the aforementioned gap is addressed by presenting a multithreading-based algorithm. This algorithm is designed for the computation of high-order DRP coefficients. The proposed approach restructures the recursive and combinatorial components of the Racah formulation to exploit task-level and data-level parallelism, thereby enhancing computational throughput while preserving numerical stability. Specifically, this framework is applied to three distinct algorithms categorized according to the DRP parameter values (a , b , α , and β):

- Zero-valued parameter strategies, specifically categorized into the balanced (ZPBA) and unbalanced (ZPUA) algorithms;
- A valued-parameter algorithm (VPA) where the parameters a , b , α , and β have non-zero values.

By integrating concurrency-aware design principles with optimized recurrence evaluation for these three algorithms, the proposed framework achieves significant acceleration compared to the best sequential implementations presented in [18]. Consequently, this makes higher-order Racah polynomials computationally feasible for large-scale scientific and engineering applications.

The organization of this paper is as follows: Section 2 introduces the related works in this field. Section 3 defines the Preliminaries of the proposed work, while Section 4 presents the proposed multithreaded algorithm to compute the DRPs. Section 5 discusses the exper-

imental results and the analysis performed to evaluate the performance of the proposed algorithm. Finally, Section 6 presents a summary of the paper, and the key findings.

2. Related Works

In this section, the state-of-the-art in DRP coefficient computation methods are summarized. The evolution is highlighted from foundational recurrences to modern stability-enhanced methods, as well as precision-focused methods, which are used to ensure numerical stability.

The fundamental computational framework for DRPs was first presented by Zhu et al. [6]. In that work, they presented two primary recurrence relations for the computation of DRP coefficients. The first recurrence relation utilizes a three-term recurrence relation in the direction of the polynomial order n . The second used the three-term recurrence relation in the signal (spatial) index s . These algorithms provide the basic concept of the mathematical formulation for most recent implementations. However, they are sensitive to initial conditions. In practice, it has been shown that the initial values reported results in divergence while computing the DRP coefficients. Thus, this has necessitated the utilization of different approach to find a new starting initial value. This, as a result, lead to use of the recurrence relation in both directions (order-based and index-based) to ensure numerical stability across all the computed DRP coefficients [18].

To address the issues of the numerical instability and overflow occur at high-order polynomial calculations, Daoui et al. [26] introduced an algorithm that is designed to stabilize the computation of the coefficients. The major improvement of this approach is the specialized recursive calculation of initial values at the boundary of the domain. This tackles the problem of the exponential growth of error at the start of the recurrence. Furthermore, the approach incorporate a stabilizing condition. When the ratio between successive polynomial values exceeds a predefined threshold, this is a sign of propagation of numerical errors. To prevent this, the computed values are forced to be set to zero. This modification is used to keep the computation of the coefficients stable by eliminating excessively large values that are not mathematically meaningful, which distort the polynomial basis.

The Gram–Schmidt Orthogonalization Process (GSOP) [27], which extends the standard recurrence-based generation, is often utilized as a tool for enhancing the precision of pre-calculated polynomials [28]. Within this framework, the GSOP acts as a corrective layer as it identifies and subtracts the projections of lower-order polynomials from the current degree to ensure orthogonality. Then, a re-normalization step is performed using a small epsilon value to prevent division by zero. Although the GSOP increases the stability of the numerical accuracy of the DRPs, it is considered a high computational bottleneck [29]. Compared with the standard recurrence relation that required $O(N^2)$ complexity, GSOP is cubically ($O(N^3)$) in terms of complexity. As a result, this makes the GSOP computationally expensive for large-scale applications where high-order polynomials are needed [30].

Recently, a new algorithm called the Improved Stabilization (ImSt) algorithm was presented [18]. This algorithm addresses the numerical instability by partitioning the DRP plane into two regions. The ImSt algorithm optimizes the stabilization conditions based on both the polynomial size and the values of the DRP parameters (a , b , α , and β). To overcome the precision issues of the computation of initial values for high-order polynomials, the ImSt algorithm utilizes the logarithmic gamma function as well as a specialized formula. This combination lead to efficient computation for a wide range of DRP parameters than earlier methods. Furthermore, this research identified a unique symmetry relation for cases when parameters are zero, which makes the algorithm use the diagonal symmetry of the DRP matrix. Comparative studies have shown that this multi-partitioning strategy significantly extends the maximum computable polynomial degree and reduces the reconstruction errors

compared to the state-of-the-art algorithms. A summary of the related works is given in Table 1.

Despite the advancements in stability-enhanced methods like ImSt and the precision-correction steps in the GSOP, a significant gap remains in the high-performance execution of these algorithms. The current literature primarily addresses numerical stability and boundary conditions, yet the most accurate methods—specifically those utilizing GSOP—suffer from an $O(N^3)$ computational complexity that becomes a bottleneck for large-scale engineering applications. Furthermore, while partitioned algorithms like ImSt improve the range of computable parameters, they are traditionally designed for sequential execution, failing to exploit the hardware concurrency available in modern multicore processors. Thus, there is a clear need for a framework that not only maintains the stability of high-order DRPs but also optimizes the task-level and data-level parallelism of the underlying recurrence relations.

Table 1. Comparative summary of the proposed algorithm with existing DRP algorithms.

Algorithm	Brief Methodology	Highlights	Limitations
Zhu et al. [6]	Three-term recurrence over order (n) and spatial index (s).	Established the mathematical foundation for weighted DRPs.	Highly sensitive to initial conditions; prone to divergence at high orders.
Daoui et al. [26]	Recursive initial value computation and ratio-based error clipping.	Introduced stability for higher parameter values; erases numerical noise.	Sequential design; clipping can lead to information loss in specific ranges.
GSOP	Post-processing through Gram–Schmidt projection and re-normalization.	Ensures maximum numerical precision and strict orthogonality.	High computational complexity $O(N^3)$; unsuitable for large N .
ImSt [18]	Asymmetric plane partitioning and Logarithmic-Gamma initialization.	Efficiently handles large polynomial sizes and specific zero-parameter cases.	Complex logic for manual partitioning; lacks hardware optimization.
Proposed Work	Parallelized recurrence with three task-specific algorithms.	Exploits multicore concurrency; optimized for balanced/unbalanced cases.	Tailored for shared-memory multicore architectures.

3. Preliminaries

In this section, the essential context of the mathematical definition of DRPs and the recurrence relation used to compute the DRP coefficients are presented.

3.1. Mathematical Definitions of DRPs

The mathematical definitions of the DRPs is presented in this section. DRPs are defined by Wilson as follows [2]:

$$\mathcal{P}_n^{(\alpha, \beta)}(\lambda(x); N) = {}_4F_3 \left(\begin{matrix} -n, \alpha + \beta + n + 1, -x, \gamma + \delta + x + 1 \\ \alpha + 1, \beta + \delta + 1, \gamma + 1 \end{matrix} \middle| 1 \right), \quad (1)$$

with ${}_4F_3(\cdot)$ being the generalized hypergeometric series, which is given by [31]:

$${}_4F_3 \left(\begin{matrix} a, b, c, d \\ e, f, g \end{matrix} \middle| z \right) = \sum_{k=0}^{\infty} \frac{(a)_k (b)_k (c)_k (d)_k}{(e)_k (f)_k (g)_k k!} (z)^k. \quad (2)$$

with $(\cdot)_k$ denotes the Pochhammer symbol, which is given by [32,33]:

$$(a)_k = a(a+1)(a+2) \cdots (a+k-1). \quad (3)$$

Zhu et al. [6] introduced a coordinate transformation by defining a new variable s such that $x = s(s+1)$. Under this transformation, the DRPs of degree n , denoted by $\mathcal{P}_n^{(\alpha,\beta)}(s; N)$, are expressed as

$$\begin{aligned} \mathcal{P}_n^{(\alpha,\beta)}(s; N) &= \\ &= \frac{1}{n!} (a+b+\alpha+1)_n (\beta+1)_n (a-b+1)_n \times \\ &\times {}_4F_3 \left(\begin{matrix} -n, a-s, a+s+1, \alpha+\beta+n+1 \\ \beta+1, a+b+\alpha+1, a-b+1 \end{matrix} \middle| 1 \right). \end{aligned} \quad (4)$$

The discrete orthogonality of these polynomials is defined by the following relation:

$$\sum_{s=a}^{b-1} \mathcal{P}_n^{(\alpha,\beta)}(s; N) \mathcal{P}_m^{(\alpha,\beta)}(s; N) \rho(s) \Delta x \left(s - \frac{1}{2} \right) = d_n^2 \delta_{nm}, \quad (5)$$

where δ_{nm} denotes the Kronecker delta [34,35], $\Delta x \left(s - \frac{1}{2} \right)$ represents the difference of the x shifted by a half, defined as $\Delta x \left(s - \frac{1}{2} \right) = \left(s + \frac{1}{2} \right) \left(s + \frac{3}{2} \right) - \left(s - \frac{1}{2} \right) \left(s + \frac{1}{2} \right) = (2s+1)$. Moreover, ρ signifies the weight function of DRPs.

$$\rho(s) = \frac{\Gamma(a+s+1)\Gamma(b+s+\alpha+1)\Gamma(b+\alpha-s)\Gamma(s-a+\beta+1)}{\Gamma(b+s+1)\Gamma(b-s)\Gamma(s-a+1)\Gamma(a-\beta+s+1)} \quad (6)$$

and d_n^2 represents the squared norm of the DRPs:

$$\begin{aligned} d_n^2 &= \frac{\Gamma(\alpha+n+1)\Gamma(\beta+n+1)\Gamma(a+b+\alpha+n+1)}{(\alpha+\beta+2n+1)\Gamma(n+1)\Gamma(b-a-n)} \times \\ &\times \frac{\Gamma(b-a+\alpha+\beta+n+1)}{\Gamma(\alpha+\beta+n+1)\Gamma(a+b-n-\beta)}. \end{aligned} \quad (7)$$

where $\Gamma(\cdot)$ represents the Gamma function [36].

It is important to note that the weighted version of the DRPs is often more suitable for numerical applications. The n th-degree weighted DRP, denoted by $\hat{\mathcal{P}}_n^{(\alpha,\beta)}(s; N)$, is defined as

$$\hat{\mathcal{P}}_n^{(\alpha,\beta)}(s; N) = \mathcal{P}_n^{(\alpha,\beta)}(s; N) \sqrt{\frac{\rho(s)}{d_n^2}} \cdot \Delta x \left(s - \frac{1}{2} \right). \quad (8)$$

To simplify the notation, we adopt $\hat{\mathcal{P}}_n^{(\alpha,\beta)}(s; N) = \hat{\mathcal{P}}_n(s)$ constraint $b = a + N$ for all subsequent derivations.

3.2. Recurrence Relations of DRPs

In this section, the recurrence relation used in the computation of the DRP coefficients is presented. Based on the previous studies presented in the Section 2, the recurrence relation is considered to have numerical accuracy and enhanced numerical stability compared with existing works. Therefore, it is employed in our proposed multithreading algorithms.

The adoption of the initial value location as well as its computation is substantial because the DRP coefficients are reliant on it. First, the location of the initial value is set at

$s = a + N - 1$. Second, a formula is used to determine the initial values for a broad DRP parameters, given by [18]

$$\begin{aligned} Z = & \psi(\beta+N) + \psi(2a+N) + \psi(\alpha+\beta+2) + \psi(2a+2N+\alpha) - (\psi(\beta+1) + \\ & \psi(\alpha+\beta+N+1) + \psi(2a+N+\alpha+1) + \psi(2a+2N-1)) \\ \hat{\mathcal{P}}_0(a+N-1) = & \exp(Z/2), \end{aligned} \quad (9)$$

where $\psi(\cdot)$ denotes the logarithmic gamma function and is defined as $\psi(x) = \log(\Gamma(x))$.

After the first initial value is computed, the two sets $\hat{\mathcal{P}}_0(s)$ and $\hat{\mathcal{P}}_1(s)$ are now computed. These values are determined by utilizing the two-term recurrence algorithm. They are calculated as follows: the formula to compute the values of $\hat{\mathcal{P}}_0(s)$ is

$$\begin{aligned} \hat{\mathcal{P}}_0(s) = & \sqrt{\frac{(a-s-1)(b+\alpha-s-1)(b+s+1)(a-\beta+s+1)(2s+1)}{(b-s-1)(2s+3)(a-\beta-s-1)(a+s+1)(b+\alpha+s+1)}} \times \hat{\mathcal{P}}_0(s+1), \\ s = & a+N-2, a+N-3, \dots, a. \end{aligned} \quad (10)$$

Then, the values of $\hat{\mathcal{P}}_1(s)$, in the second row, are computed based on $\hat{\mathcal{P}}_0(s)$ as follows:

$$\begin{aligned} \hat{\mathcal{P}}_1(s) = & -\left(\left((-a+b-1)\alpha+b^2-s^2-a-s-1\right)\beta + \right. \\ & \left. (a^2-s^2+b-s-1)\alpha + a^2+b^2-2(s^2+s)-1\right) \times \\ & \sqrt{\frac{\alpha+\beta+3}{(a-b+1)(a+b-1-\beta)(\alpha+1)(\beta+1)(-b-\alpha+a-\beta-1)}} \times \\ & \sqrt{\frac{1}{(b+\alpha+a+1)}} \hat{\mathcal{P}}_1(s+1), \\ s = & a, a+1, \dots, a+N-1. \end{aligned} \quad (11)$$

Based on the methodology reported in [18], controlling indices are introduced to improve the numerical stability of the DRP coefficient computation. The procedure begins by calculating the coefficients $\hat{\mathcal{P}}_n(a)$ and $\hat{\mathcal{P}}_n(a+N-1)$ in the first and last columns, respectively. The locations of the peak values are then determined. The coefficients associated with $\hat{\mathcal{P}}_n(a)$ are subsequently computed using a two-term recurrence relation.

The $\hat{\mathcal{P}}_n(a)$ is computed as follows:

$$\begin{aligned} \hat{\mathcal{P}}_{n+1}(a) = & \\ & -\sqrt{\frac{(N-n-1)(\alpha+\beta+2n+3)(\alpha+\beta+n+1)}{(2a+N-\beta-n-1)(\alpha+\beta+2n+1)(\alpha+n+1)}} \times \\ & \times \sqrt{\frac{(\beta+n+1)(2a+N+\alpha+n+1)}{(N+\alpha+\beta+n+1)(n+1)}} \hat{\mathcal{P}}_n(a), \\ n = & 1, 2, \dots, N-2. \end{aligned} \quad (12)$$

And $\hat{\mathcal{P}}_n(a + N - 1)$ is computed as follows:

$$\begin{aligned} \hat{\mathcal{P}}_{n+1}(a + N - 1) = & \sqrt{\frac{(N-n-1)(\alpha+\beta+2n+3)(\alpha+\beta+n+1)}{(n+1)(2a+N+\alpha+n+1)(\alpha+\beta+2n+1)}} \times \\ & \times \sqrt{\frac{(\alpha+n+1)(2a+N-\beta-n-1)}{(N+\alpha+\beta+n+1)(n+1)}} \hat{\mathcal{P}}_n(a + N - 1), \\ & n = 1, 2, \dots, N - 2. \end{aligned} \quad (13)$$

The index that correspond to the peak value in the first column is found by

$$ind_0 = \arg \max_{n=0,1,\dots,N-1} \hat{\mathcal{P}}_n(a), \quad (14)$$

while the peak value in the last column is found by

$$ind_{N-1} = \arg \max_{n=0,1,\dots,N-1} \hat{\mathcal{P}}_n(a + N - 1). \quad (15)$$

The boundary separating the parts is defined by a line which connects the points (a, ind_0) and $(a + N - 1, ind_{N-1})$.

The coefficient values of DRPs in the first part are generated by utilizing the three-term recurrence relation in the n -direction as follows:

$$\hat{\mathcal{P}}_n(s) = \Theta_1 \hat{\mathcal{P}}_{n-1}(s) + \Theta_2 \hat{\mathcal{P}}_{n-2}(s), \quad (16)$$

The recurrence relation in first part is carried out for $s = a, a + 1, \dots, a + N - 2$ and $n = 2, 3, \dots, ind_s - 1$. It is noteworthy that the border between the first and second parts is located at index ind_s , which is given by

$$ind_s = \text{round} \left(ind_0 + \frac{ind_{N-1} - ind_0}{N - 1} (s - a) \right), \quad (17)$$

The coefficient values of the DRP in Part 2 are computed by utilizing (16). Note that, after the computation of each value, the following stabilizing condition is used for each order n :

$$\hat{\mathcal{P}}_n(s) = 0 \text{ if } |\hat{\mathcal{P}}_n(s)| < 10^{-5} \text{ and } |\hat{\mathcal{P}}_n(s)| > |\hat{\mathcal{P}}_{n-1}(s)|, \quad (18)$$

The recurrence relation in Part 2 is applied in the range $s = a, a + 1, \dots, a + N - 2$ and $n = ind_s, ind_s + 1, \dots, N - 1$.

It is noteworthy that a special case where the DRPs parameters $a = \alpha = \beta = 0$, $\left(\hat{\mathcal{P}}_n^{(0,0)}(s) \right)$ is also used. In this special case, the coefficients are calculated in the range $n = 0, 1, \dots, N - 1$ and $s = n, n + 1, \dots, N - 1$ as follows:

$$\hat{\mathcal{P}}_n(s) = \Theta_{10} \hat{\mathcal{P}}_{n-1}(s) + \Theta_{20} \hat{\mathcal{P}}_{n-2}(s), \quad (19)$$

the remaining coefficients values are calculated by utilizing the symmetry relation as follows:

$$\hat{\mathcal{P}}_s^{(0,0)}(n) = (-1)^{(s-n)} \hat{\mathcal{P}}_n^{(0,0)}(s), \quad (20)$$

The recurrence coefficients Θ_2 , Θ_{10} , and Θ_{20} appear in (16) and (19) are defined in [18].

4. Methodology

In this section, the proposed multithreaded algorithms to compute the DRPs are presented. There are four cases presented: unbalanced DRPs with zero-values parameters, balanced DRPs with zero-value parameters, unbalanced DRPs with given parameter values, balanced DRPs with given parameter values.

Multithread processing, also called multithreading, is a computing technique that partitions a program into multiple threads to facilitate concurrent processing within a single task. A thread is the smallest unit of execution within a process that can be scheduled and managed independently by an operating system [37,38]. By enabling these units to operate independently, the approach optimizes the utilization of CPU resources, particularly within multicore architectures [39].

4.1. Methodology of the Zero-Parameter Unbalanced Algorithm (ZPUA)

Let N denote the DRP matrix dimension ($N \times N$ matrix), and M the required number of threads to process the polynomial coefficients. The total number of columns to be processed is denoted by C_{total} , whereas T represents the effective number of threads utilized, where $T \leq M$. The workload distribution is determined by the base allocation B and the remainder Δ_c , where Δ_c represents the columns that cannot be evenly partitioned among the threads. Let $t \in \{0, 1, \dots, T-1\}$ denote the thread index. For each thread t , the assigned number of columns is denoted by C_t . The corresponding starting and ending column indices are given by S_t and E_t respectively. E_t is treated as an exclusive upper bound such that the assigned columns span $[S_t, E_t)$. The total number of columns to be computed ranges from index 2 to $N-2$. Because the upper bound $N-1$ is exclusive, the total column count is given by

$$C_{\text{total}} = \max(0, (N-1) - 2) = N-3, \quad \text{for } N \geq 3. \quad (21)$$

To prevent the allocation of idle threads, the effective number of active threads T is bounded by the total number of available columns:

$$T = \max(1, \min(M, C_{\text{total}})). \quad (22)$$

To partition the workload, the total number of columns is divided evenly among the active threads, establishing the base allocation B and remainder Δ_c :

$$B = \left\lfloor \frac{C_{\text{total}}}{T} \right\rfloor, \quad (23)$$

$$\Delta_c = C_{\text{total}} \bmod T. \quad (24)$$

To distribute this remainder uniformly, the first Δ_c threads are assigned exactly one additional column. Consequently, the total number of columns C_t assigned to a specific thread t is given by the piecewise function:

$$C_t = \begin{cases} B+1, & \text{if } t < \Delta_c \\ B, & \text{if } t \geq \Delta_c. \end{cases} \quad (25)$$

The index ranges assigned to each thread are allocated sequentially. The starting column index S_t for thread t is offset from the global starting index of 2, yielding

$$S_t = \begin{cases} 2 + t(B+1), & \text{if } t < \Delta_c \\ 2 + tB + \Delta_c, & \text{if } t \geq \Delta_c. \end{cases} \quad (26)$$

The corresponding ending column index E_t , representing the exclusive upper bound of the assigned block, is simply

$$E_t = S_t + C_t. \quad (27)$$

For more clarification, a representation of the workload partitioning is depicted in Figure 1. This figure depicts the assignment of columns across the DRP matrix, which demonstrates that the first Δ_c threads receive slightly larger blocks of size $B + 1$ compared with the remaining threads, which receive blocks of size B . This will ensure complete, non-overlapping coverage of the computational domain without requiring dynamic synchronization.

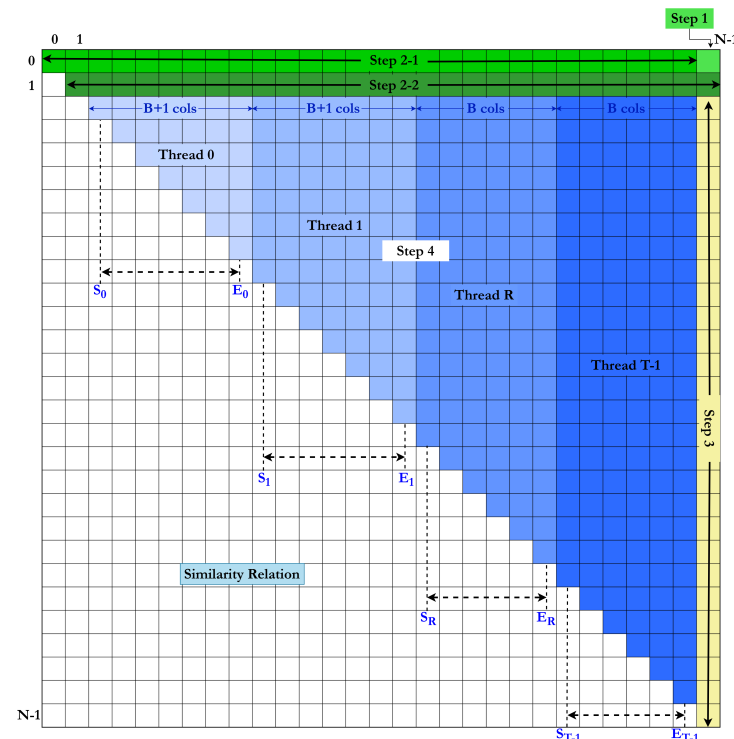


Figure 1. Distribution of the computational workload across threads for the unbalanced zero-parameter DRP. Step 1 computes the initial value. Steps 2-1 and 2-2 compute the initial sets at the first and second rows, respectively. Step 3 computes the coefficients for the last column.

Triangular Row Computation and Dynamic Early Stopping

For any assigned column $s \in [S_t, E_t)$, the row-level calculation is bounded by the diagonal, iterating exclusively over the upper triangular portion of the matrix where the row index $n \in \{2, 3, \dots, s\}$. To optimize the computational cost across the columns, the dynamic early-stopping criterion previously established in (18) is applied.

The polynomial recurrence relation defined in (19) is computed sequentially for $n \leq s$. If the calculated value $\hat{\mathcal{P}}_n^{(0,0)}(s)$ falls below the tolerance threshold (10^{-5}) and subsequently exhibits growth in magnitude relative to the previous row (indicating a lack of convergence), the computation for that column is immediately terminated. Therefore, the final number of rows evaluated for a given column s is bounded by the stopping row $n_{\text{stop}}(s)$, mathematically defined as

$$n_{\text{stop}}(s) = \begin{cases} \min\{n \mid n \leq s, (|\hat{\mathcal{P}}_n^{(0,0)}(s)| \leq 10^{-5} \wedge |\hat{\mathcal{P}}_n^{(0,0)}(s)| > |\hat{\mathcal{P}}_{n-1}^{(0,0)}(s)|)\}, \\ s \quad \text{otherwise.} \end{cases} \quad (28)$$

Furthermore, rather than requiring a separate computational phase, the corresponding elements of the lower triangle are evaluated directly within each thread alongside their upper-triangular counterparts. For each element $\hat{\mathcal{P}}_n^{(0,0)}(s)$ calculated in the upper triangle, the symmetric element ($n > s$) is populated simultaneously using the parity-based similarity relation defined in (20). Algorithm 1 summarizes the workflow of the proposed multithreaded ZPUA algorithm.

Algorithm 1 Methodology of the zero-parameter unbalanced algorithm (ZPUA)

Require: Matrix dimension N , Maximum requested threads M

Ensure: Populated DRP Coefficient Matrix under zero-parameter configuration

// STAGE 1: HOST INITIALIZATION & COLUMN ALLOCATION

```

1:  $C_{\text{total}} \leftarrow \max(0, N - 3)$ 
2: if  $C_{\text{total}} == 0$  then
3:   return ▷ Terminate case execution
4: end if
5:  $T \leftarrow \max(1, \min(M, C_{\text{total}}))$ 
6:  $B \leftarrow \left\lfloor \frac{C_{\text{total}}}{T} \right\rfloor$ 
7:  $\Delta_c \leftarrow C_{\text{total}} \bmod T$ 

```

// STAGE 2: CONCURRENT THREAD PROCESSING

```

8: spawn  $T$  threads concurrently for each thread index  $t \in \{0, 1, \dots, T - 1\}$  do
9:   if  $t < \Delta_c$  then
10:     $C_t \leftarrow B + 1$ 
11:     $S_t \leftarrow 2 + t(B + 1)$ 
12:   else
13:     $C_t \leftarrow B$ 
14:     $S_t \leftarrow 2 + tB + \Delta_c$ 
15:   end if
16:  $E_t \leftarrow S_t + C_t$ 
17: for  $s = S_t$  to  $E_t - 1$  do ▷ Process assigned column block
18:   for  $n = 2$  to  $s$  do ▷ Iterate upper triangular matrix
19:     $\text{DRP}[n][s] \leftarrow \text{Evaluate\_Recurrence\_Relation}(n, s)$ 
20:    if  $n > 2$  then ▷ Apply dynamic early stopping check
21:      if  $|\text{DRP}[n][s]| \leq 10^{-5} \wedge |\text{DRP}[n][s]| > |\text{DRP}[n - 1][s]|$  then
22:        break ▷ Terminate current column processing early
23:      end if
24:    end if
25:     $\text{DRP}[s][n] \leftarrow \text{Apply\_Symmetry}(\text{DRP}[n][s], n, s)$ 
26:   end for
27: end for
28: end spawn
29: synchronize / join all  $T$  threads
30: return DRP Matrix

```

4.2. Methodology of the Zero-Parameter Balanced Algorithm (ZPBA)

This section formalizes the static, symmetrically balanced workload distribution method. The computation for the zero-parameter DRP matrix is restricted to the upper-triangular region ($n \leq s$). Consequently, the number of computational operations grows linearly with the index s . To achieve load balancing without costly synchronization overhead, a mirrored block partitioning strategy is employed.

Let N denote the DRP matrix dimension, and M the required number of threads. The interior columns to be processed range from $s = 2$ to $N - 2$. The total number of columns to be computed is

$$C_{\text{total}} = \max(0, (N - 1) - 2) = N - 3, \quad \text{for } N \geq 3. \quad (29)$$

Let $T = \max(1, \min(M, C_{\text{total}}))$ be the effective number of active threads. To balance the linearly increasing workload, the computational domain is bisected into a left half and a right half:

$$C_{\text{left}} = \left\lfloor \frac{C_{\text{total}}}{2} \right\rfloor + (C_{\text{total}} \bmod 2), \quad C_{\text{right}} = \left\lfloor \frac{C_{\text{total}}}{2} \right\rfloor. \quad (30)$$

Each thread $t \in \{0, 1, \dots, T - 1\}$ is assigned exactly two disjoint blocks of columns: one from the left half ($\mathcal{C}_{t,\text{left}}$) and one from the right half ($\mathcal{C}_{t,\text{right}}$).

The left half is partitioned sequentially using a base allocation $B_{\text{left}} = \lfloor C_{\text{left}}/T \rfloor$ and remainder $\Delta_{\text{left}} = C_{\text{left}} \bmod T$. The number of columns $C_{t,\text{left}}$ assigned to thread t is

$$C_{t,\text{left}} = B_{\text{left}} + \begin{cases} 1, & \text{if } t < \Delta_{\text{left}} \\ 0, & \text{otherwise.} \end{cases} \quad (31)$$

The starting and exclusive ending column indices for the left block are given by

$$S_{t,\text{left}} = 2 + tB_{\text{left}} + \min(t, \Delta_{\text{left}}), \quad (32)$$

$$E_{t,\text{left}} = S_{t,\text{left}} + C_{t,\text{left}}. \quad (33)$$

To counteract the increasing computational weight of the columns, the assignment of blocks in the right half is *mirrored* (reversed). We define the reversed thread index as $\bar{t} = T - 1 - t$.

The right half is partitioned using $B_{\text{right}} = \lfloor C_{\text{right}}/T \rfloor$ and $\Delta_{\text{right}} = C_{\text{right}} \bmod T$. The number of columns $C_{t,\text{right}}$ assigned to thread t depends on its reversed index:

$$C_{t,\text{right}} = B_{\text{right}} + \begin{cases} 1, & \text{if } \bar{t} < \Delta_{\text{right}} \\ 0, & \text{otherwise.} \end{cases} \quad (34)$$

The starting and exclusive ending column indices for the right block are shifted by the size of the left half and assigned according to $\bar{t}$:

$$S_{t,\text{right}} = 2 + C_{\text{left}} + \bar{t}B_{\text{right}} + \min(\bar{t}, \Delta_{\text{right}}), \quad (35)$$

$$E_{t,\text{right}} = S_{t,\text{right}} + C_{t,\text{right}}. \quad (36)$$

The total subset of columns processed by thread t , denoted as $\mathcal{C}_t$, is the union of its left and right assigned blocks, defined by the half-open intervals:

$$\mathcal{C}_t = [S_{t,\text{left}}, E_{t,\text{left}}) \cup [S_{t,\text{right}}, E_{t,\text{right}}). \quad (37)$$

This ensures that a thread processing the “lightest” columns on the left simultaneously processes the “heaviest” columns on the right, providing an optimal static load balance. The resulting mirrored block assignment and perfectly balanced static computational volume per thread are illustrated in Figure 2.

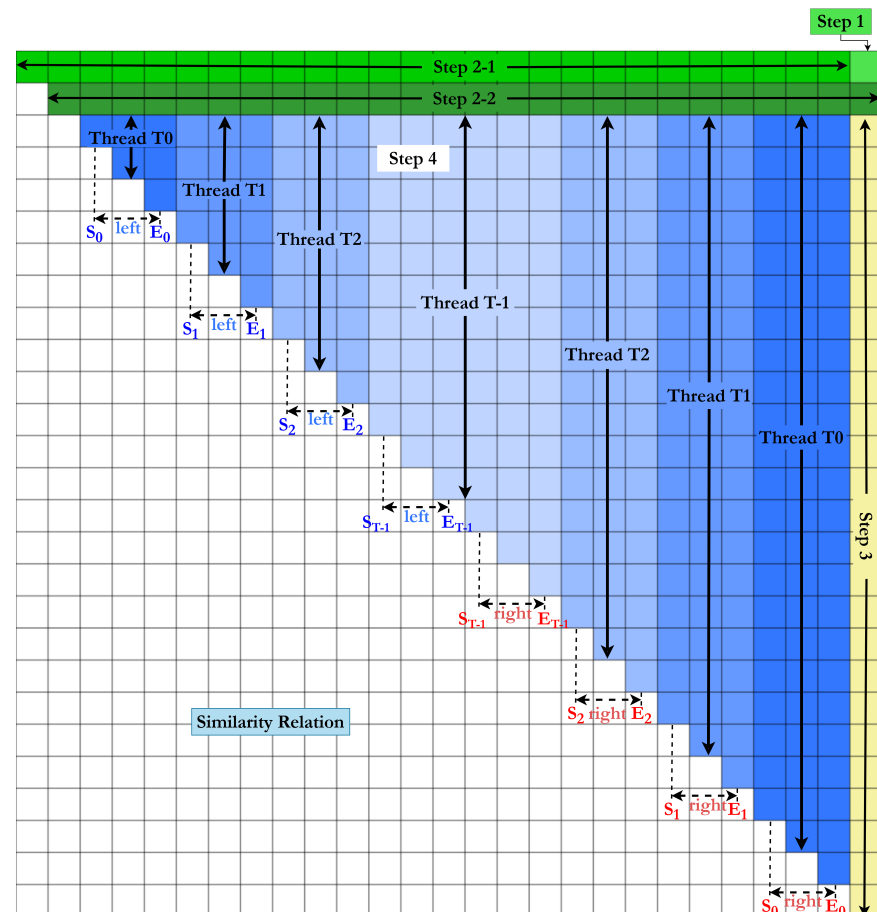


Figure 2. Distribution of the computational workload across threads for the balanced zero-parameter DRP. Step 1 computes the initial value. Steps 2-1 and 2-2 compute the initial sets at the first and second rows, respectively. Step 3 computes the coefficients for the last column.

Triangular Row Computation and Early Stopping

For any column $s \in \mathcal{C}_t$ assigned to thread t , the row-level calculation is bounded by the diagonal, iterating exclusively over the upper-triangular portion where $n \in \{2, 3, \dots, s\}$.

To optimize computational cost, the dynamic early-stopping criterion previously established in (18) is applied. The polynomial recurrence relation defined in (19) is computed sequentially for $n \leq s$. If the calculated value $\hat{\mathcal{P}}_n^{(0,0)}(s)$ falls below the tolerance threshold (10^{-5}) and subsequently exhibits growth in magnitude relative to the previous row (indicating a lack of convergence), the computation for that column is terminated.

The final number of rows evaluated for a given column s is bounded by $n_{\text{stop}}(s)$:

$$n_{\text{stop}}(s) = \begin{cases} \min\{n \mid n \leq s, (|\hat{\mathcal{P}}_n^{(0,0)}(s)| \leq 10^{-5} \wedge |\hat{\mathcal{P}}_n^{(0,0)}(s)| > |\hat{\mathcal{P}}_{n-1}^{(0,0)}(s)|)\}, \\ s \quad \text{otherwise.} \end{cases} \quad (38)$$

Furthermore, the corresponding elements of the lower triangle are evaluated directly within each thread alongside their upper-triangular counterparts. For each element $\hat{\mathcal{P}}_n^{(0,0)}(s)$ calculated in the upper triangle, the symmetric element ($n > s$) is populated simultaneously using the parity-based similarity relation defined in (20). Figure 3 illustrates the workload computational volume for the balanced zero-parameter DRP. Algorithm 2 summarizes the workflow of the proposed multithreaded ZPBA algorithm.

Algorithm 2 Methodology of the zero-parameter balanced algorithm (ZPBA)**Require:** Matrix dimension N , Maximum requested threads M **Ensure:** Populated DRP Coefficient Matrix under zero-parameter configuration

```

// STAGE 1: HOST INITIALIZATION & LOAD BALANCING
1:  $C_{\text{total}} \leftarrow \max(0, N - 3)$ 
2: if  $C_{\text{total}} == 0$  then
3:   return ▷ Terminate case execution
4: end if
5:  $T \leftarrow \max(1, \min(M, C_{\text{total}}))$ 
6:  $C_{\text{left}} \leftarrow \left\lfloor \frac{C_{\text{total}}}{2} \right\rfloor + (C_{\text{total}} \bmod 2)$ 
7:  $C_{\text{right}} \leftarrow \left\lfloor \frac{C_{\text{total}}}{2} \right\rfloor$ 
8:  $B_{\text{left}} \leftarrow \left\lfloor \frac{C_{\text{left}}}{T} \right\rfloor, \Delta_{\text{left}} \leftarrow C_{\text{left}} \bmod T$ 
9:  $B_{\text{right}} \leftarrow \left\lfloor \frac{C_{\text{right}}}{T} \right\rfloor, \Delta_{\text{right}} \leftarrow C_{\text{right}} \bmod T$ 

// STAGE 2: CONCURRENT THREAD PROCESSING
10: spawn  $T$  threads concurrently for each thread index  $t \in \{0, 1, \dots, T - 1\}$  do
11:    $\bar{t} \leftarrow T - 1 - t$  ▷ Define mirrored index for the right half
   // Left block allocation calculations
12: if  $t < \Delta_{\text{left}}$  then
13:    $C_{t,\text{left}} \leftarrow B_{\text{left}} + 1$ 
14: else
15:    $C_{t,\text{left}} \leftarrow B_{\text{left}}$ 
16: end if
17:  $S_{t,\text{left}} \leftarrow 2 + tB_{\text{left}} + \min(t, \Delta_{\text{left}})$ 
18:  $E_{t,\text{left}} \leftarrow S_{t,\text{left}} + C_{t,\text{left}}$ 
   // Right block allocation calculations
19: if  $\bar{t} < \Delta_{\text{right}}$  then
20:    $C_{t,\text{right}} \leftarrow B_{\text{right}} + 1$ 
21: else
22:    $C_{t,\text{right}} \leftarrow B_{\text{right}}$ 
23: end if
24:  $S_{t,\text{right}} \leftarrow 2 + C_{\text{left}} + \bar{t}B_{\text{right}} + \min(\bar{t}, \Delta_{\text{right}})$ 
25:  $E_{t,\text{right}} \leftarrow S_{t,\text{right}} + C_{t,\text{right}}$ 
   // Mirrored processing execution loop
26: for each interval  $[S, E] \in \{[S_{t,\text{left}}, E_{t,\text{left}}], [S_{t,\text{right}}, E_{t,\text{right}}]\}$  do
27:   for  $s = S$  to  $E - 1$  do
28:     for  $n = 2$  to  $s$  do ▷ Iterate upper triangular portion
29:        $\text{DRP}[n][s] \leftarrow \text{Evaluate\_Recurrence\_Relation}(n, s)$ 
30:       if  $n > 2$  then ▷ Apply dynamic early stopping check
31:         if  $|\text{DRP}[n][s]| \leq 10^{-5} \wedge |\text{DRP}[n][s]| > |\text{DRP}[n-1][s]|$  then
32:           break ▷ Terminate current column processing early
33:         end if
34:       end if
35:        $\text{DRP}[s][n] \leftarrow \text{Apply\_Parity\_Symmetry}(\text{DRP}[n][s], n, s)$ 
36:     end for
37:   end for
38: end for
39: end spawn
40: synchronize / join all  $T$  threads
41: return DRP Matrix

```

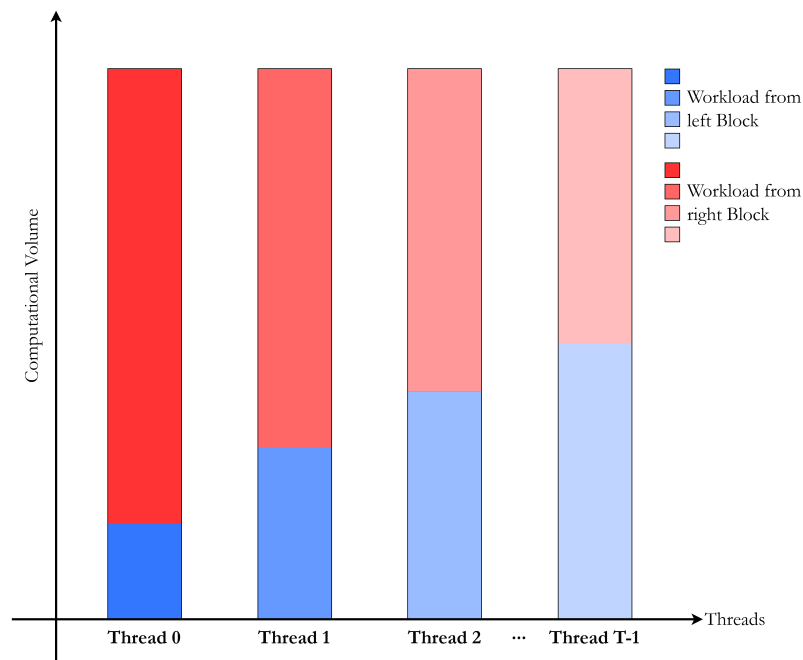


Figure 3. Workload computational volume for balanced zero-parameter DRP.

4.3. Methodology of the Valued-Parameter Algorithm (VPA)

This section formalizes the static partitioning of the workload into independent blocks for processing. This approach eliminates the overhead of costly synchronization by pre-determining the column distribution, though it may result in variable execution times per thread due to the varying row depths.

4.3.1. Deterministic Column Partitioning

In this approach, $C_{\text{total}} = N - 2$ and $T = \max(1, \min(M, C_{\text{total}}))$. The total number of columns is divided evenly among the active threads, establishing a base allocation B and remainder Δ_c :

$$B = \left\lfloor \frac{C_{\text{total}}}{T} \right\rfloor, \quad \Delta_c = C_{\text{total}} \bmod T. \quad (39)$$

To uniformly distribute the remainder, the first Δ_c threads receive exactly one additional column. Thus, the total number of columns C_t assigned to thread t is determined by the piecewise function:

$$C_t = \begin{cases} B + 1, & \text{if } t < \Delta_c \\ B, & \text{if } t \geq \Delta_c. \end{cases} \quad (40)$$

The starting column index at S_t for thread t is calculated directly without interthread communication:

$$S_t = \begin{cases} 1 + t(B + 1), & \text{if } t < \Delta_c \\ 1 + tB + \Delta_c, & \text{if } t \geq \Delta_c. \end{cases} \quad (41)$$

The exclusive ending column index E_t is given by $E_t = S_t + C_t$. This guarantees that the subset of columns $\mathcal{C}_t$, processed by thread t , is strictly contiguous and disjoint from all other threads, defined by the half-open interval:

$$\mathcal{C}_t = [S_t, E_t). \quad (42)$$

Figure 4 illustrates this piecewise distribution of the computational workload across threads for the valued-parameters DRP matrix. It demonstrates how the first Δ_c threads

receive slightly larger blocks while maintaining strict contiguity without dynamic synchronization.

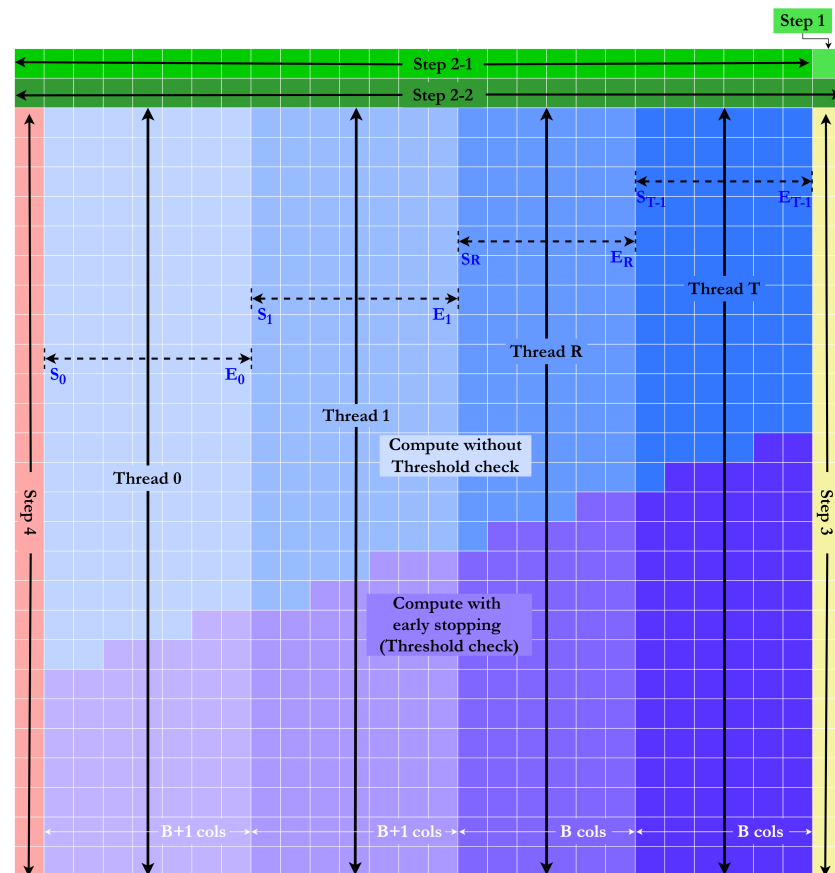


Figure 4. Distribution of the computational workload across threads for the valued-parameter DRP. Step 1 computes the initial value. Steps 2-1 and 2-2 compute the initial sets at the first and second rows, respectively. Step 3 computes the coefficients for the last column. Step 4 computes the coefficients for the first column.

4.3.2. Row Computation and Dynamic Stopping Criterion

Once the contiguous block of columns $\mathcal{C}_t$ is assigned to thread t , the row-level calculations are performed independently for each column $s \in \mathcal{C}_t$.

The algorithm utilizes the exact same dynamic early-stopping criterion defined previously. The expected peak $\text{mis}(s)$ is calculated, and the rows are evaluated unconditionally through $\text{mis}(s)$. For $n > \text{mis}(s)$, the polynomial recurrence relation is halted the moment the value falls below the tolerance threshold and ceases converging, i.e., $|\hat{\mathcal{P}}_n^{(0,0)}(s)| \leq 10^{-5}$ and $|\hat{\mathcal{P}}_n^{(0,0)}(s)| > |\hat{\mathcal{P}}_{n-1}^{(0,0)}(s)|$.

Because the number of rows evaluated, $n_{\text{stop}}(s) - 1$, varies smoothly but monotonically across the matrix dimensions, the static distribution of $\mathcal{C}_t$ columns inevitably leads to threads assigned to the left side of the matrix (lower t) executing marginally more mathematical operations than threads assigned to the right side (higher t). Algorithm 3 summarizes the workflow of the proposed multithreaded VPA algorithm.

Algorithm 3 Methodology of the valued-parameter algorithm (VPA)**Require:** Matrix dimension N , Maximum requested threads M **Ensure:** Populated DRP Coefficient Matrix under valued-parameter configuration

// STAGE 1: HOST INITIALIZATION & COLUMN PARTITIONING

1: $C_{\text{total}} \leftarrow N - 2$ 2: $T \leftarrow \max(1, \min(M, C_{\text{total}}))$ 3: $B \leftarrow \left\lfloor \frac{C_{\text{total}}}{T} \right\rfloor$ 4: $\Delta_c \leftarrow C_{\text{total}} \bmod T$

// STAGE 2: CONCURRENT THREAD PROCESSING

5: **spawn** T threads concurrently **for** each thread index $t \in \{0, 1, \dots, T - 1\}$ **do**6: **if** $t < \Delta_c$ **then**7: $C_t \leftarrow B + 1$ 8: $S_t \leftarrow 1 + t(B + 1)$ 9: **else**10: $C_t \leftarrow B$ 11: $S_t \leftarrow 1 + tB + \Delta_c$ 12: **end if**13: $E_t \leftarrow S_t + C_t$ 14: **for** $s = S_t$ **to** $E_t - 1$ **do**▷ Process contiguous column block C_t 15: $\text{mis}(s) \leftarrow \text{Calculate_Expected_Peak}(s)$ 16: **for** $n = 1$ **to** $N - 1$ **do**

▷ Evaluate row-level calculations independently

17: $\text{DRP}[n][s] \leftarrow \text{Evaluate_Valued_Recurrence}(n, s)$ 18: **if** $n > \text{mis}(s)$ **then**

▷ Apply dynamic early-stopping check past peak

19: **if** $|\text{DRP}[n][s]| \leq 10^{-5} \wedge |\text{DRP}[n][s]| > |\text{DRP}[n-1][s]|$ **then**20: **break**

▷ Terminate current column processing early

21: **end if**22: **end if**23: **end for**24: **end for**25: **end spawn**26: **synchronize / join** all T threads27: **return** DRP Matrix**5. Experimental Results and Discussion**

In this section, a comprehensive experimental analysis is performed to evaluate the performance of the proposed algorithm. In addition, the proposed algorithm is benchmarked in terms of the execution time, thread synchronization, and CPU affinity utilization. Furthermore, the proposed algorithm is evaluated and compared to existing works. The results were obtained by conducting the experiments using an Intel Core i7-12700KF CPU (Intel Corporation, Santa Clara, CA, USA) (8 P-cores and 4 E-cores, which provide 20 logical CPUs), 32 GB of RAM, and Microsoft Windows runtime library threading. To ensure statistical reliability and account for runtime fluctuations, all of the following experiments were performed for 20 independent runs, and the average execution times were computed.

The performance is benchmarked across varying polynomial sizes ($N = 1000$ to 8000), threads (2, 4, 8, 16, 32, 64, 128, and 256), and CPU affinities (2, 4, 8, 16, and 20). It is noteworthy that, as the proposed methods parallelize the state-of-the-art sequential DRP algorithm without altering its mathematical formulation or the strict sequence of row-level operations, the computed coefficients retain the exact same numerical accuracy, round-off behavior, and stability characteristics as the original implementation.

For $N = 1000$, Figure 5 shows that parallelization costs easily dominate the limited workload. ZPBA and ZPUA achieve the lowest execution times—with ZPBA performing best—while VPA remains the slowest. As core count increases, execution time optimizes at approximately 0.577 msec on 16 and 20 logical CPUs. However, severe oversubscription

starting at 128 threads degrades performance to roughly 11 msec. This shows that for small workloads, the scheduling and synchronization overheads completely dominate parallel scaling benefits. As a results, using threads more than 16 is not beneficial and the optimal thread selection dropped to zero beyond that point.

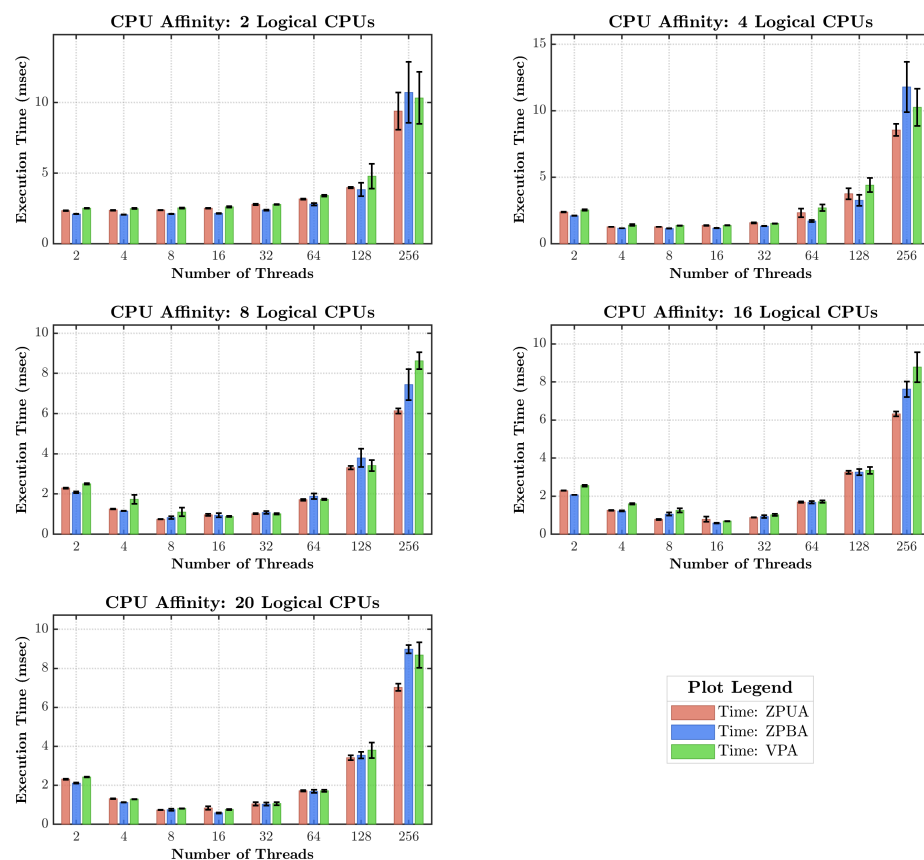


Figure 5. Computation time for polynomial size $N = 1000$.

Figure 6 shows the computation time for polynomials size $N = 2000$. The results depict an improved computation-to-overhead ratio, which represents better hardware scaling. ZPBA and ZPUA continue to dominate, while VPA remains the slowest (noting that ZPBA achieves the lowest execution times). At this polynomial size ($N = 2000$), with a larger workload, it shows effective utilization of more physical cores. This drives execution time down to an optimal 1.42 msec on 20 logical CPUs using 16 threads. On the other hand, a severe oversubscription cost persists; scaling from 128 threads degrades performance to between 3.52 msec and 12.45 msec depending on logical CPU affinity. Furthermore, near-optimal thread selection peaks precisely at 16 threads and become negligible beyond 32. This proves that large hyperthreading remains inefficient at this size.

For $N = 4000$, Figure 7 illustrates a highly efficient computation-to-overhead ratio, which makes it excellent hardware scaling. ZPBA consistently achieves the lowest execution times, and outperforms VPA. The increased workload efficiently utilizes up to 16 logical CPUs and reduces the execution time near-linearly from 30.78 msec on 2 logical CPUs to 5.20 msec on 16 logical CPUs. It is noteworthy that sufficient computational density shifts the optimal concurrency point up to 32 threads. Despite this increased workload, with a large number of threads, oversubscription remains adverse to performance. The system manages 128 or 256 threads, and this degrades the performance across all logical CPUs affinities and increases execution times over 11.2 msec on the 20-logical-CPU setup. This demonstrates that limiting how many threads are created to the hardware's effective capacity (32 threads) is critical for optimal performance.

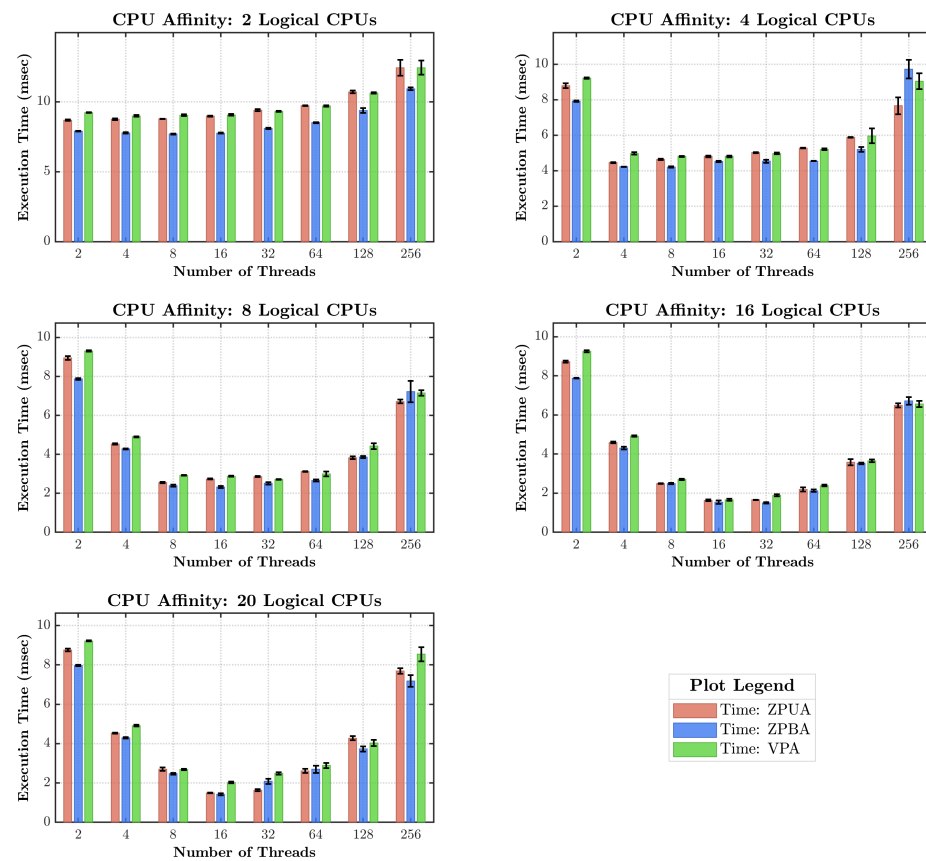


Figure 6. Computation time for polynomial size $N = 2000$.

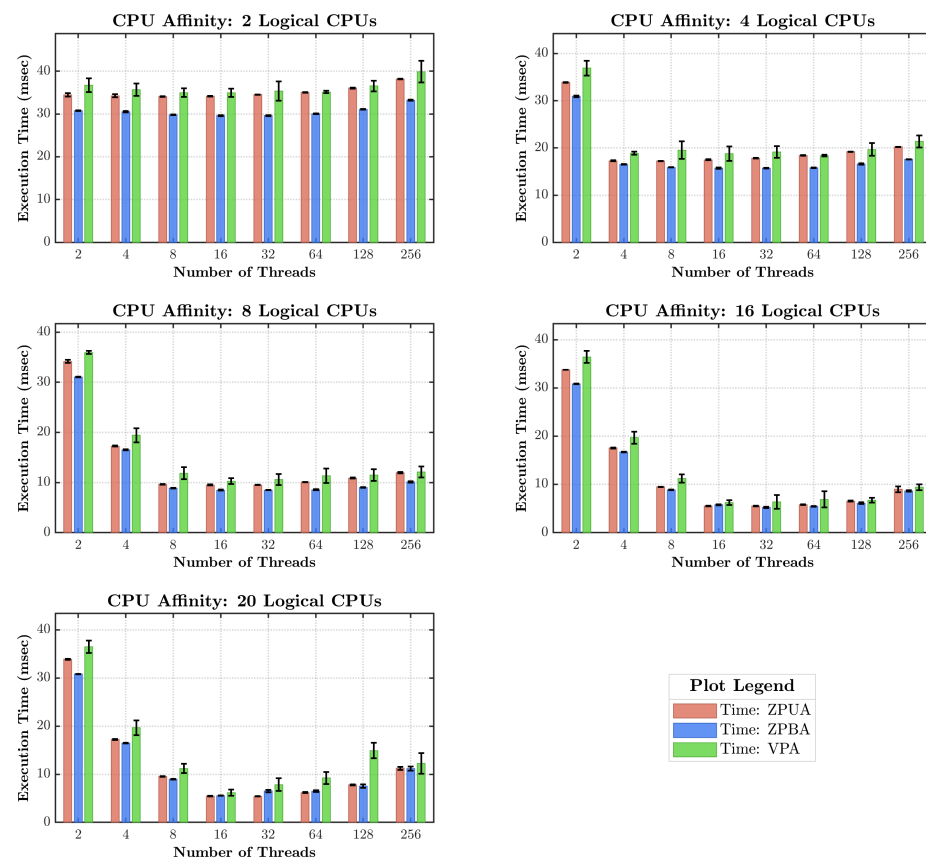


Figure 7. Computation time for polynomial size $N = 4000$.

For the largest tested polynomial size ($N = 8000$), Figure 8 depicts an excellent strong scaling. ZPBA consistently outperforms ZPUA and VPA, which achieves the lowest execution times. At this workload, the execution time is reduced to near-linear from over 135 msec to an optimal 20.7 msec on 20 logical CPUs using 16 threads. High computational density largely masks the thread management overheads (such as scheduling and coordination overhead), which becomes less noticeable. Oversubscribing to 128 or 256 threads results in a slight increase in the execution time, which reaches ≈ 27.6 msec, showing a relatively mild performance degradation and confirming a compute-constrained state. Nevertheless, limiting the number of simultaneously running threads between 16 and 32 threads remains optimal.

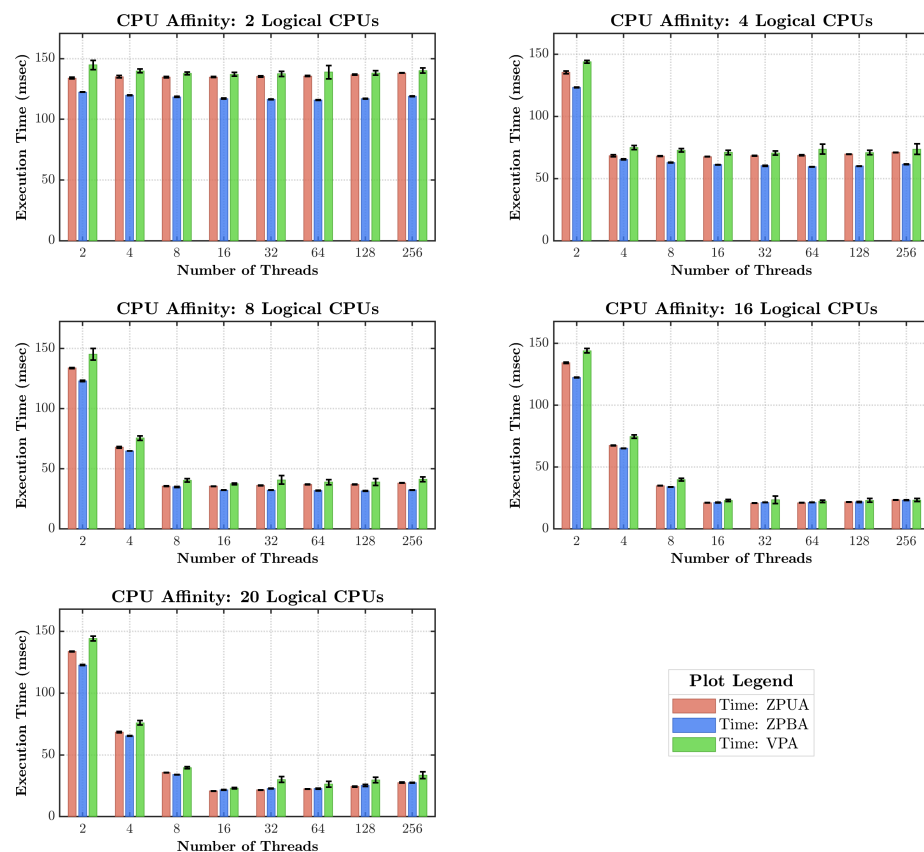


Figure 8. Computation time for polynomial size $N = 8000$.

To analyze the effectiveness and scalability of the presented algorithms, an evaluation of performance improvements over the sequential baseline [18] is conducted. Since no published parallel implementations for DRPs are currently available, the serial implementation of the same DRP algorithm was selected as the baseline. This ensures that the observed performance gains are solely attributable to the proposed parallelization strategy without introducing algorithmic differences. The following analysis investigates the effect of changes in polynomial size (N) on the balance between parallel execution gains and runtime overhead.

For small polynomial size of $N = 1000$, the comparison with the sequential baseline shows high sensitivity to parallel scheduling overhead. As shown in Figure 9, ZPBA consistently delivers the best performance, with a peak speedup is $6.55\times$. However, this small workload results in an inefficient computation-to-overhead ratio where oversubscription to a large number of threads is negatively impact the performance. The management of 128 or 256 threads drops the speedup below unity to a minimum of $0.32\times$ for ZPUA on 4 logical CPUs. At this low polynomial size, the overhead of thread creation, frequent

context switching, and cache thrashing completely overwhelms the theoretical benefits of parallelism. This surely makes excessive concurrency less efficient than a single-threaded implementation [18].

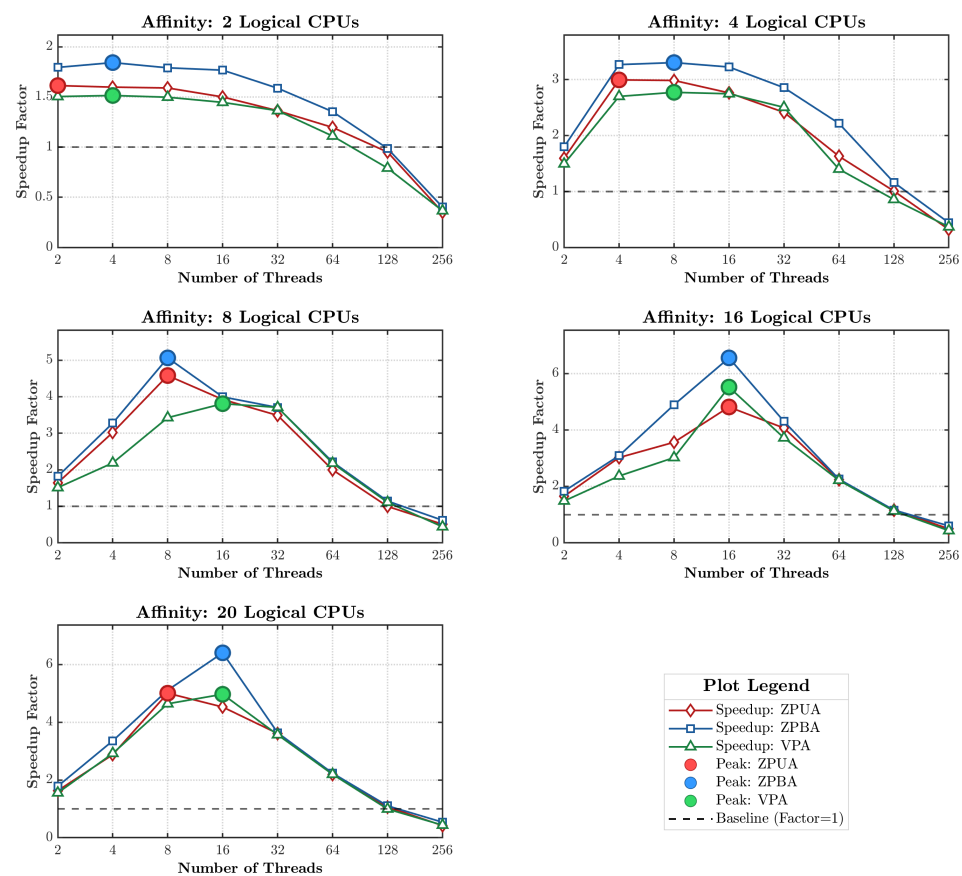


Figure 9. Performance improvement for $N = 1000$ for different thread and logical CPU configurations.

For $N = 2000$, the performance improvements become clear as the polynomial size increased which allows the system to better distribute scheduling overhead. As depicted in Figure 10, ZPBA outperforms ZPUA and VPA, which reaches a peak speedup factor of $\approx 10.11 \times$ on the 16-logical-CPU configuration. While thread oversubscription at 256 threads still results in performance degradation, the impact on the speedup is markedly less compared with polynomials size of $N = 1000$. This confirms that higher computational demands better justify the overhead of parallelization. This successfully transitions the framework from a scheduling-constrained state to the one that effectively utilizes available hardware concurrency.

As the polynomial size increases to $N = 4000$, the algorithm transitions into a compute-intensive phase in which the substantial workload far exceeds parallel scheduling overhead. As shown in Figure 11, ZPBA translates its high efficiency into an impressive maximum speedup factor of $10.91 \times$ over the sequential baseline. Furthermore, a shift in scaling behavior is observed regarding massive thread oversubscription. Unlike the impact of degradation occurred at smaller sizes, managing 256 threads at $N = 4000$ never degrades performance below the unthreaded baseline. Even in the highly constrained 2-logical-CPU setup, the 256-thread execution maintains a positive speedup factor of $1.71 \times$ for ZPBA. This approves that at this scale, the theoretical benefits of multithreading fully realized [18].

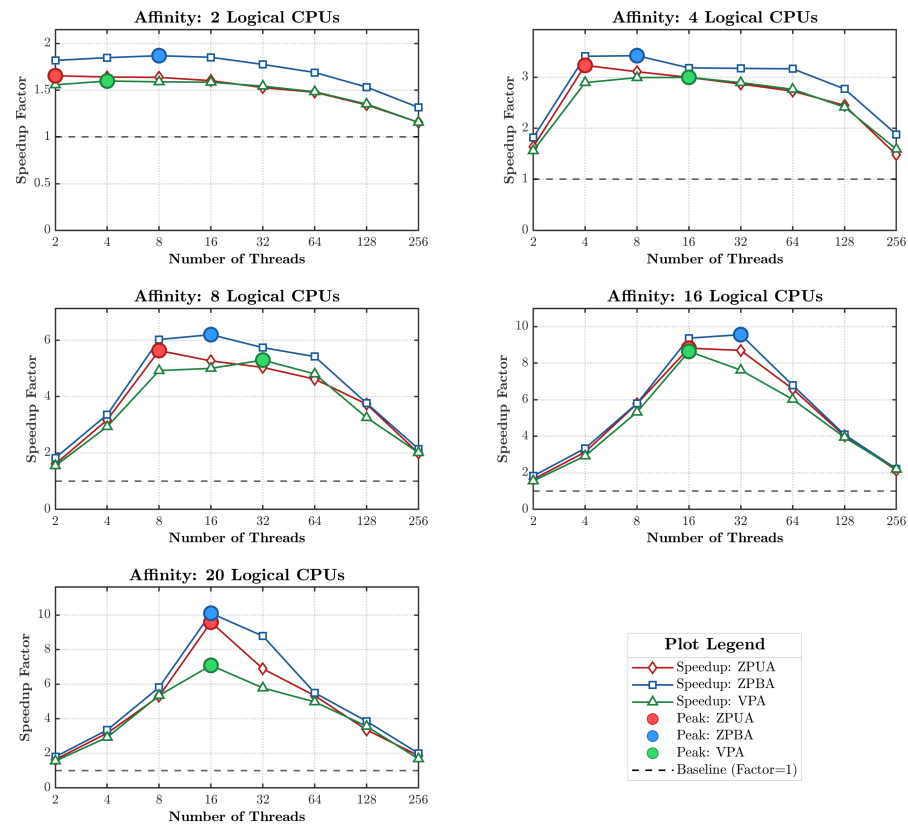


Figure 10. Performance improvement for $N = 2000$ for different thread and logical CPU configurations.

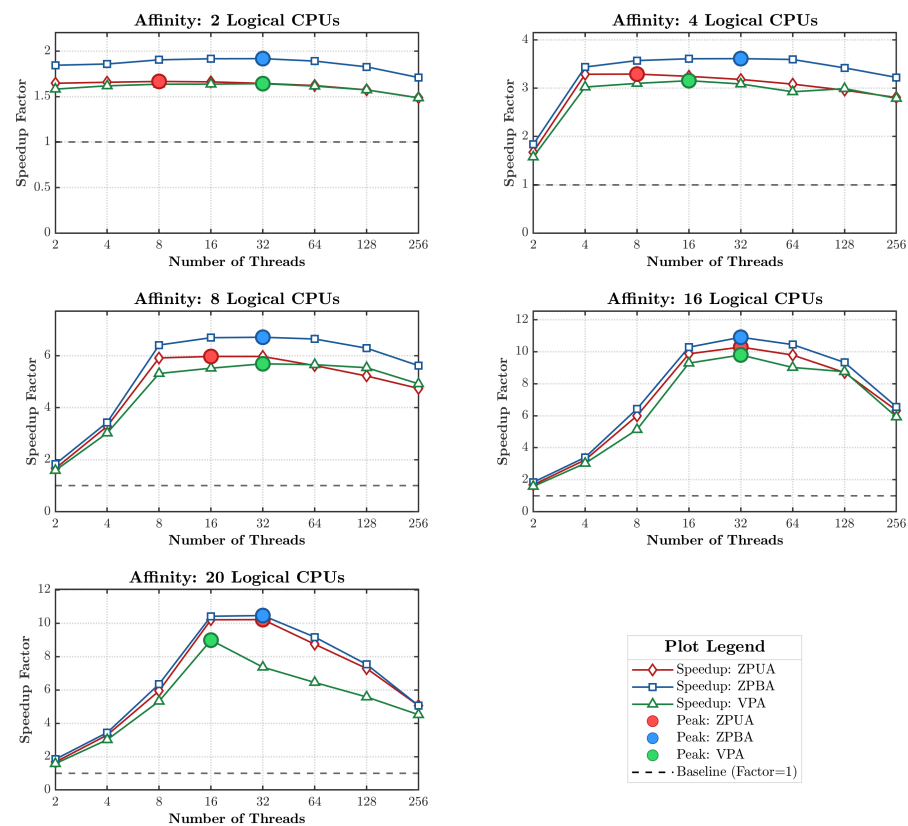


Figure 11. Performance improvement for $N = 4000$ for different thread and logical CPU configurations.

Expanding the analysis to the highest tested polynomial size of $N = 8000$, the framework operates consistently within a compute-bound regime where the immense computa-

tional payload completely dominates overheads (see Figure 12). Consistent with previous observations, ZPBA dictates optimal efficiency, corresponding to a maximum speedup factor of $10.44\times$ over the baseline [18]. At this large scale, the algorithm demonstrates an extraordinary resilience to massive thread oversubscription. Even under the most extreme constraint—managing 256 threads on a mere 2 logical CPUs—ZPBA maintains a strong positive speedup of $1.87\times$, while the 16-logical-CPU configuration processing 256 threads still delivers a massive $9.61\times$ speedup. This robust scaling behavior verifies that for highly dense computational problems, maximizing concurrency strictly benefits throughput and entirely neutralizes the adverse context-switching effects that penalized smaller problem sizes.

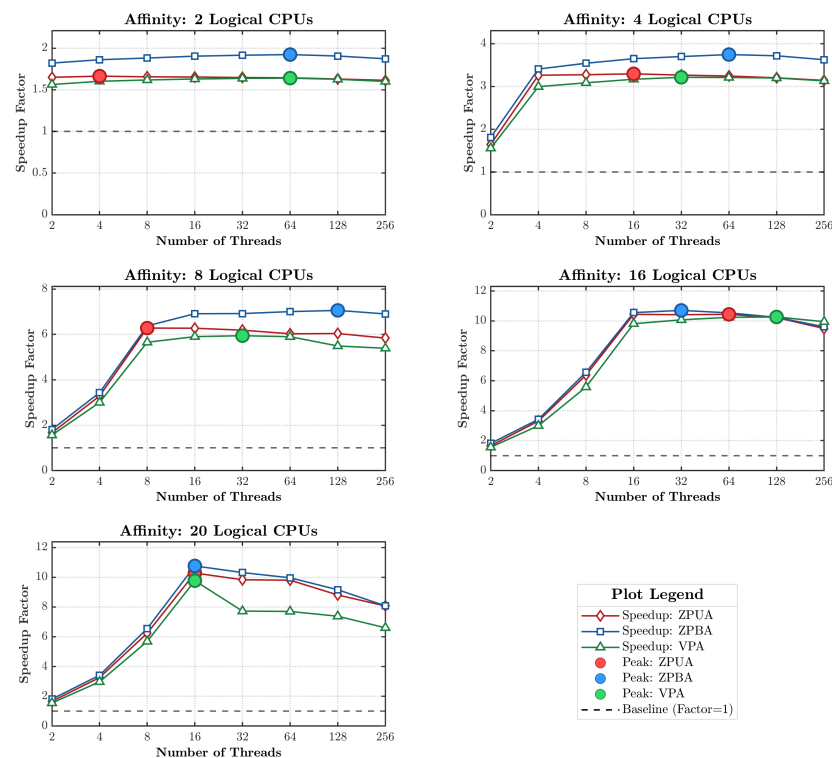


Figure 12. Performance improvement for $N = 8000$ for different thread and logical CPU configurations.

To analyze the scaling behavior of the unbalanced algorithm across varying workload densities, an examination of the optimal thread count mapping depicted in Figure 13 reveals highly non-linear characteristics dependent on both hardware concurrency and problem size. For smaller polynomial sizes of $N = 1000$ and $N = 2000$, the optimal thread count generally mirrors the available logical CPUs up to the 16-logical-CPU configuration at a 1:1 ratio, whereas the 20-logical-CPU setup introduces scheduling inefficiencies that force the optimal count to drop to 8 or stall at 16 threads. A scaling pivot occurs at $N = 4000$, where the increased computational payload justifies thread oversubscription, causing optimal counts to jump significantly across all setups, including a peak of 32 threads for both the 16-logical-CPU and 20-logical-CPU affinities. At the highest workload densities of $N = 8000$, a stark divergence emerges: the 16-logical-CPU configuration successfully leverages massive concurrency to reach an absolute peak of 64 optimal threads at $N = 8000$, while the 20-logical-CPU setup severely plateaus at just 16 threads. This behavior indicates that the unbalanced workload distribution strategy inherently struggles with synchronization or load distribution across the 20-logical-CPU topology, resulting in diminishing returns and dictating a heavily constrained optimal thread ceiling to prevent performance degradation at extreme problem sizes.

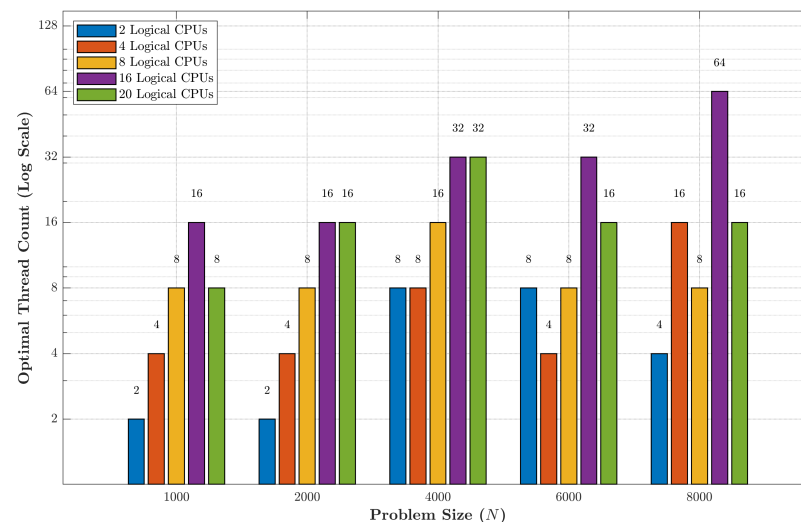


Figure 13. Optimal thread count distribution across varying polynomial sizes (N) and logical CPU configurations for the ZPUA algorithm.

An examination of the optimal thread count mapping (shown in Figure 14) for the balanced algorithm reveals a distinctly different progression profile compared to the unbalanced approach. For lower computational workloads at $N = 1000$ and $N = 2000$, the balanced strategy scales relatively proportionally to the hardware, though it exhibits a readiness to embrace thread oversubscription earlier, optimizing at 32 threads on the 16-logical-CPU configuration at $N = 2000$. The most striking behavioral anomaly occurs at the $N = 4000$ scaling pivot, where the optimal thread count universally locks at exactly 32 threads across all multi-logical-CPU setups ranging from 2 to 20 logical CPUs. This uniform stabilization suggests an ideal equilibrium point where the balanced workload distribution perfectly synchronizes the payload irrespective of the underlying logical CPU topology. However, as the problem size expands to extreme densities at $N = 8000$, a sharp architectural divide emerges. Fewer-logical-CPU configurations (two, four, and eight logical CPUs) aggressively rely on massive oversubscription to maintain throughput, reaching an absolute peak of 128 optimal threads on eight logical CPUs at $N = 8000$. In stark contrast, higher-tier architectures face severe diminishing returns; the 16-logical-CPU setup regresses to optimize at 32 threads, and the 20-logical-CPU setup is heavily bottlenecked, plateauing at a mere 16 threads. This indicates that while the balanced algorithm achieves remarkable synchronization at medium workloads, its distribution overhead at extreme problem sizes creates critical limitations on high-logical CPU topologies, strictly capping their viable concurrency while smaller architectures freely oversubscribe to mask latency.

Analyzing the optimal thread count mapping (illustrated in Figure 15) for the parameterized algorithm exposes a highly erratic and non-linear scaling profile that heavily contrasts with the more predictable behaviors of the unbalanced and balanced strategies. At lower computational densities ($N = 1000$ and $N = 2000$), the algorithm demonstrates a tendency to plateau early on higher-logical-CPU architectures, stubbornly optimizing at exactly 16 threads for the 20-logical-CPU setup, and similarly stalling at 16 threads on the 16-logical-CPU setup at $N = 1000$. As the workload transitions through the $N = 4000$ intermediate sizes, the optimal thread mapping exhibits significant oscillation; for instance, at $N = 4000$, the two-logical-CPU configuration aggressively oversubscribes to 32 threads, while the four-logical-CPU setup paradoxically regresses to 16 threads. At the maximum evaluated density of $N = 8000$, this volatility culminates in massive, isolated spikes, such as the 16-logical-CPU configuration heavily oversubscribing to an absolute peak of 128 optimal threads. Notably, throughout all tested problem sizes, the 20-logical-CPU topology

remains severely bottlenecked, consistently stalling at a maximum of 16 optimal threads regardless of the computational payload. This pronounced inconsistency suggests that the specific tuning mechanisms inherent to the parameterized approach are highly sensitive to the underlying hardware topology, resulting in unstable load distribution and unpredictable optimal concurrency limits across varying problem sizes.

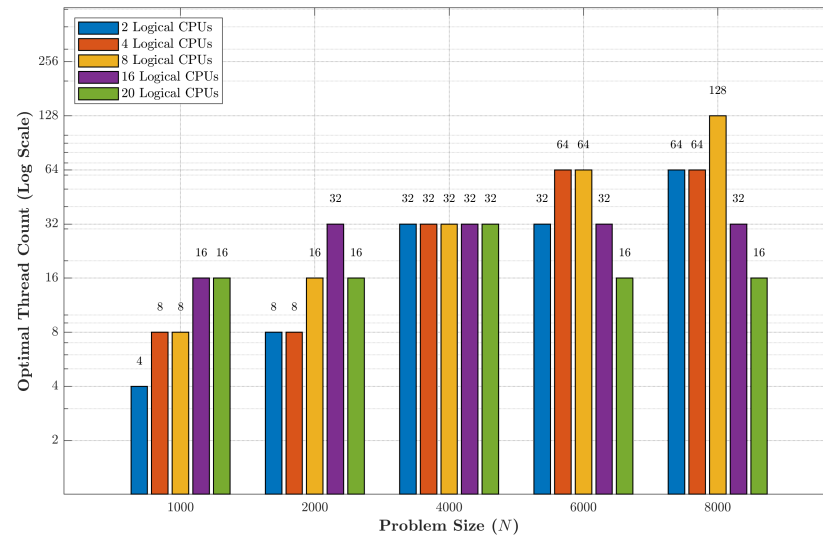


Figure 14. Optimal thread count distribution across varying polynomial sizes (N) and logical CPU configurations for the ZPBA algorithm.

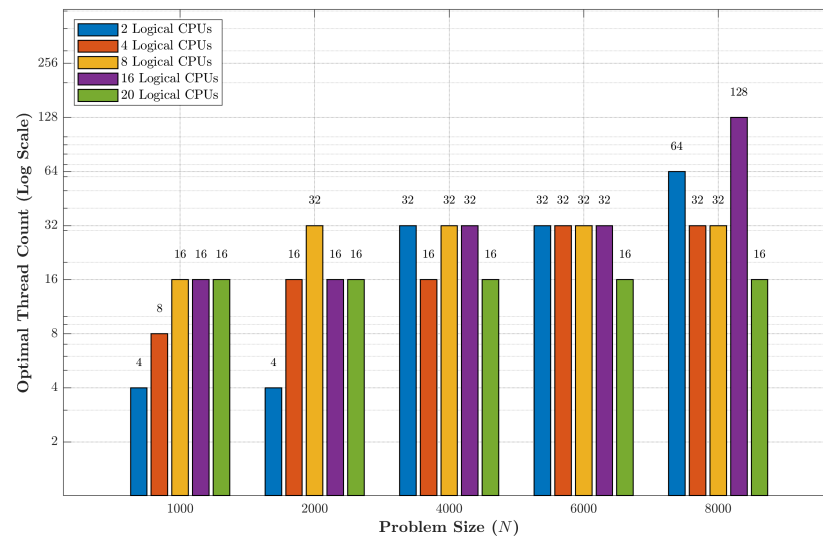


Figure 15. Optimal thread count distribution across varying polynomial sizes (N) and logical CPU configurations for the VPA algorithm.

Overall, the experimental results indicate that the optimal thread count is generally 556 achieved at ≈ 16 threads (Figure 16a). This behavior is primarily attributed to the hybrid architecture of the Intel Core i7-12700KF processor, which consists of eight performance (P) cores supporting hyperthreading (16 logical CPUs) and four efficient (E) cores, resulting in a total of 20 logical CPUs. The proposed DRP computation is compute-intensive and therefore benefits mainly from the high-performance P-cores. As a result, increasing the thread count beyond 16 provides only limited additional computational capability while introducing additional scheduling, synchronization, and memory-access overheads associated with the efficient cores. Although workloads with larger polynomial sizes can effectively utilize up to 32 threads in some cases, excessive oversubscription (128 or

256 threads) consistently results in performance degradation, confirming that thread counts close to the hardware's effective processing capacity provide the best balance between parallelism and runtime overhead.

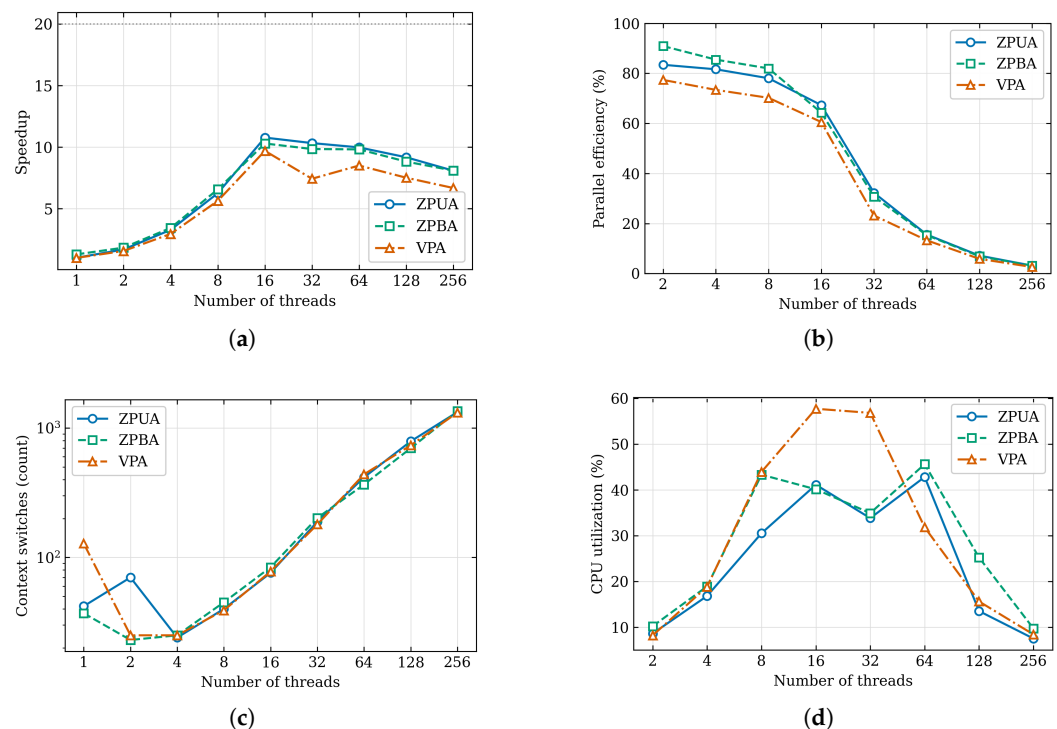


Figure 16. Scalability at $N = 8000$ on the 20-logical-CPU i7-12700KF: (a) speedup versus thread count, (b) parallel efficiency versus thread count, (c) Context-switch overhead versus thread count, and (d) CPU utilization versus thread count.

The trends are quantitatively supported by the recorded metrics (Figure 16): at $N = 8000$, the parallel efficiency is maximal at ≈ 60 – 67% at ≈ 16 threads (Figure 16b), and is reduced to $\approx 3\%$ at 256 threads, while the context switches increase by a factor of ~ 10 ($\approx 80 \rightarrow \approx 1350$) (Figure 16c), the thread creation/join cost increases from <1 ms to 5–10 ms, and the CPU utilization of the assigned logical CPUs decreases from ≈ 40 – 58% to $<10\%$ (Figure 16d). This confirms that, outside the effective logical CPU count, the execution is dominated by scheduling and synchronization overhead, and limited by shared resources such as memory bandwidth, rather than by useful computation.

To sum up, the experimental results demonstrate that the proposed multithreading algorithms substantially improve the computational performance of high-order DRPs. Notably, the ZPBA algorithm consistently achieves the best scalability and execution efficiency, while the ZPUA and VPA algorithms exhibit greater sensitivity to workload characteristics and hardware topology.

6. Conclusions

This paper presented a novel multithreading framework to overcome the historical sequential bottlenecks associated with computing high-order DRPs. By carefully mapping the mathematical recurrence relations of DRPs into concurrent tasks, we introduced three distinct partitioning strategies: ZPBA, ZPUA, and VPA. The experimental evaluation demonstrated that transitioning from sequential to parallel execution dramatically reduces computation time, achieving speedups exceeding $10\times$ for large matrix dimensions. Among the proposed strategies, ZPBA proved to be the most robust and highly scalable. By employing a mirrored block distribution, ZPBA perfectly balanced the uniform compu-

tational payload of zero-valued parameters, allowing for maximum hardware utilization. Conversely, the evaluation highlighted the inherent complexities of non-zero parameter evaluations; specifically, VPA is subject to varying row depths and load imbalances, requiring precise thread tuning to achieve optimal scalability on modern multicore architectures. Ultimately, this framework bridges the gap between theoretical mathematical models and high-performance computing, rendering the generation of large-scale DRP matrices computationally viable for advanced digital signal processing and quantum physics applications. Future work will investigate GPU-based implementations of the proposed algorithms using many-core parallel computing frameworks, such as CUDA or OpenCL, to further accelerate the computation of high-order DRPs and to evaluate their performance relative to the multithreaded CPU implementations presented in this work.

Author Contributions: Methodology, R.S.M., H.S.R. and H.J.M.; software, R.S.M., B.M.M. and S.H.A.; validation, A.H.A.-s., M.A., M.S.A. and S.H.A.; formal analysis, R.S.M., H.S.R. and A.H.A.-s.; investigation, B.M.M., A.H.A.-s., H.J.M. and M.A.; writing—original draft preparation, R.S.M., H.J.M., A.H.A.-s., B.M.M. and M.S.A.; writing—review and editing, H.S.R., H.J.M., S.H.A., M.S.A. and M.A.; visualization, H.S.R., B.M.M. and M.A.; supervision, S.H.A.; project administration, S.H.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank the University of Baghdad for their general support.

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

Abbreviations

The following abbreviations are used in this manuscript:

CPU	Central processing unit
DRPs	Discrete Racah polynomials
GSOP	Gram–Schmidt orthogonalization process
VPA	Valued-parameter algorithm
ZPBA	Zero-parameter balanced algorithm
ZPUA	Zero-parameter unbalanced algorithm

References

1. Anderson, R.W.; Aquilanti, V.; Bitencourt, A.C.P.; Marinelli, D.; Ragni, M. The Screen Representation of Spin Networks: 2D Recurrence, Eigenvalue Equation for $6j$ Symbols, Geometric Interpretation and Hamiltonian Dynamics. In *Computational Science and Its Applications—ICCSA 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 46–59. https://doi.org/10.1007/978-3-642-39643-4_4.
2. Wilson, J.A. Hypergeometric Series Recurrence Relations and Some New Orthogonal Functions. Ph.D. Thesis, University of Wisconsin, Madison, WI, USA, 1978.
3. Chen, Q.; Murakami, J. Asymptotics of quantum $6j$ symbols. *J. Differ. Geom.* **2023**, *123*, 1–20.
4. Koekoek, R.; Lesky, P.A.; Swarttouw, R.F. *Hypergeometric Orthogonal Polynomials and Their Q-Analogues*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2010.
5. Abdalhussain, S.H.; Mahmmod, B.M.; Flusser, J.; AL-Utaibi, K.A.; Sait, S.M. Fast overlapping block processing algorithm for feature extraction. *Symmetry* **2022**, *14*, 715.

6. Zhu, H.; Shu, H.; Liang, J.; Luo, L.; Coatrieux, J.L. Image analysis by discrete orthogonal Racah moments. *Signal Process.* **2007**, *87*, 687–708. <https://doi.org/10.1016/j.sigpro.2006.07.007>.
7. See, K.W.; Loke, K.S.; Lee, P.A.; Loe, K.F. Image reconstruction using various discrete orthogonal polynomials in comparison with DCT. *Appl. Math. Comput.* **2007**, *193*, 346–359. <https://doi.org/10.1016/j.amc.2007.03.080>.
8. Kalpana, J.; Krishnamoorthi, R. Color image retrieval technique with local features based on orthogonal polynomials model and SIFT. *Multimed. Tools Appl.* **2016**, *75*, 49–69. <https://doi.org/10.1007/s11042-014-2262-1>.
9. Abdul-Hadi, A.M.; Mohammed, A.A.; Mohammed, H.J.; Muhammad, R.S.; Alwhelat, A.; Alsabah, M.; Mahmmoud, B.M.; Abdulhussain, S.H. A toolbox for real orthogonal polynomials. *SoftwareX* **2026**, *33*, 102545. <https://doi.org/10.1016/j.softx.2026.102545>.
10. Kririm, S.; Zouhri, A.; Aboukassim, A.; Arjda, E.H.; Mallahi, M.E. Advancing 3D Object Estimation with Innovative Racah Moments and Robust Deconvolution Filter. *Circuits Syst. Signal Process.* **2025**, *45*, 5506–5545.
11. Chekira, C.; Marzouq, M.; El Fadili, H.; Lakhliat, Z.; da Graça Ruano, M. Join security and block watermarking-based evolutionary algorithm and Racah moments for medical imaging. *Biomed. Signal Process. Control* **2024**, *96*, 106554. <https://doi.org/10.1016/j.bspc.2024.106554>.
12. Chekira, C.; El-Amarty, N.; Hosni, M.; El Fadili, H. A watermarking-based multimodal healthcare data protection using blockchain and deep learning. *Biomed. Signal Process. Control* **2026**, *120*, 110021. <https://doi.org/10.1016/j.bspc.2026.110021>.
13. Karmouni, H.; Tahiri, M.A.; Dagal, I.; Amakdouf, H.; Jamil, M.O.; Qjidaa, H.; Sayyouri, M. Secure and optimized satellite image sharing based on chaotic $e\pi$ map and Racah moments. *Expert Syst. Appl.* **2024**, *236*, 121247. <https://doi.org/10.1016/j.eswa.2023.121247>.
14. El-Khanchouli, K.; Mansouri, H.; El Ghouate, N.; Karmouni, H.; Joudar, N.E.; Sayyouri, M.; Askar, S.S.; Abouhawwash, M. Protecting medical images using a zero-watermarking approach based on fractional Racah moments. *IEEE Access* **2025**, *13*, 16978–17001.
15. Benouini, R.; Batioua, I.; Zenkouar, K.; Zahi, A.; El Fadili, H.; Qjidaa, H. Fast and accurate computation of Racah moment invariants for image classification. *Pattern Recognit.* **2019**, *91*, 100–110.
16. El Alami, A.; Mesbah, A.; Berrahou, N.; Berrahou, A.; Qjidaa, H. Quaternion discrete Racah moments convolutional neural network for color face recognition. In Proceedings of the 2022 International Conference on Intelligent Systems and Computer Vision (ISCV), Fez, Morocco, 18–20 May 2022; 2022; pp. 1–5.
17. Aymane, B.; Khalid, Z.; Azeddine, Z. Discrete Orthogonal Moments-Based Deep Clustering: A Racah Polynomials and Cauchy Distribution Approach. In Proceedings of the 2025 11th International Conference on Optimization and Applications (ICOA), Dakhla, Morocco, 29–30 October 2026; 2025; pp. 1–6.
18. Mahmmoud, B.M.; Abdulhussain, S.H.; Suk, T.; Alsabah, M.; Hussain, A. Accelerated and Improved Stabilization for High Order Moments of Racah Polynomials. *IEEE Access* **2023**, *11*, 110502–110521. <https://doi.org/10.1109/ACCESS.2023.3321969>.
19. Munir, A.; Gordon-Ross, A.; Ranka, S.; Koushanfar, F. A queueing theoretic approach for performance evaluation of low-power multi-core embedded systems. *J. Parallel Distrib. Comput.* **2014**, *74*, 1872–1890. <https://doi.org/10.1016/j.jpdc.2013.07.003>.
20. Yu, M.; Zhou, Y.; Liu, H.; Liu, B. High-Performance Parallel Direct Georeferencing for Massive ULS LiDAR Measurements. *Remote Sens.* **2026**, *18*, 949.
21. Mochurad, L.; Tsmots, I.; Mostova, V.; Kystsiv, K. Multi-Level Parallel CPU Execution Method for Accelerated Portion-Based Variant Call Format Data Processing. *Computation* **2026**, *14*, 48.
22. Chapman, B.; Jost, G.; Van Der Pas, R. *Using OpenMP: Portable Shared Memory Parallel Programming*; MIT Press: Cambridge, MA, USA, 2007.
23. Bouknight, W.J. Thread-Level Parallelism. In *Computer Architecture: A Quantitative Approach*; Elsevier: Amsterdam, The Netherlands, 2025; p. 379.
24. Cheng, J.R.; Gen, M. Accelerating genetic algorithms with GPU computing: A selective overview. *Comput. Ind. Eng.* **2019**, *128*, 514–525.
25. Flayyih, W.N.; Al-sudani, A.H.; Mahmmoud, B.M.; Abdulhussain, S.H.; Alsabah, M. High-Performance Krawtchouk Polynomials of High Order Based on Multithreading. *Computation* **2024**, *12*, 115. <https://doi.org/10.3390/computation12060115>.
26. Daoui, A.; Karmouni, H.; Sayyouri, M.; Qjidaa, H. Stable Analysis of Large-Size Signals and Images by Racah's Discrete Orthogonal Moments. *J. Comput. Appl. Math.* **2022**, *403*, 113830.
27. Dax, A. A modified Gram–Schmidt algorithm with iterative orthogonalization and column pivoting. *Linear Algebra Appl.* **2000**, *310*, 25–42.
28. García-Ortiz, A.; Murgan, T.; Glesner, M. Moment-based estimation of switching activity for correlated distributions. In *Proceedings of the International Workshop on Power and Timing Modeling, Optimization and Simulation, Santorini, Greece, 15–17 September 2004*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 859–868.
29. Balabanov, O.; Grigori, L. Randomized Gram–Schmidt process with application to GMRES. *SIAM J. Sci. Comput.* **2022**, *44*, A1450–A1474.

30. Golub, G.H.; Van Loan, C.F. *Matrix Computations*, 3rd ed.; Johns Hopkins University Press: Baltimore, MD, USA, 1996; p. 694.
31. Wang, K.; Ping, Z.; Sheng, Y. Development of image invariant moments—A short overview. *Chin. Opt. Lett.* **2016**, *14*, 091001–91011. <https://doi.org/10.3788/col201614.091001>.
32. Abramowitz, M.; Stegun, I.A. (Eds.) *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, 9th printing ed.; Dover: New York, NY, USA, 1972.
33. Salouan, R.; Safi, S.; Bouikhalene, B. Handwritten Arabic Characters Recognition Using Methods Based on Racah, Gegenbauer, Hahn, Tchebychev and Orthogonal Fourier-Mellin Moments. *Int. J. Adv. Sci. Technol.* **2015**, *78*, 13–28.
34. Flusser, J.; Suk, T.; Zitová, B. *2D and 3D Image Analysis by Moments*; Wiley: Hoboken, NJ, USA, 2016; pp. 1–529. <https://doi.org/10.1002/9781119039402>.
35. Gray, R.M. Toeplitz and Circulant Matrices: A Review. *Found. Trends® Commun. Inf. Theory* **2006**, *2*, 155–239. <https://doi.org/10.1561/01000000006>.
36. C., B.C.; Spanier, J.; Oldham, K.B. *An Atlas of Functions*; Hemisphere Publishing Corporation: New York, NY, USA, 1987.
37. Maltare, N.; Goyani, M.; Vahora, S. *Principles of Operating System Design and Virtualization Technologies*; Deep Science Publishing: London, UK, 2025.
38. Benner, E.; Aboulhaga, A.; Patel, D. *DevSecOps in Oracle Cloud: Securing and Automating Oracle Cloud Infrastructure*; Pearson Education: London, UK, 2025.
39. Kavi, K.M.; Lee, B.; Hirson, A.R. Multithreaded systems. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 1998; Volume 46, pp. 287–328.

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.