

Exact RNS Comparison through Ordered Clusters

Parham Ghayour

`monsieur.parham.ghayour@gmail.com`

Sorbonne University <https://orcid.org/0009-0001-5607-1532>

Research Article

Keywords: residue number system; mixed-radix conversion; exact comparison; ordered partitions; workload analysis; reproducible computation.

Posted Date: October 21st, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11434280/v1>

License: © ⓘ This work is licensed under a Creative Commons Attribution 4.0 International License.
[Read Full License](#)

Additional Declarations: The authors declare no competing interests.

Exact RNS Comparison through Ordered Clusters

Mixed-Radix Structure, Optimal Tag Partitions,
Workload Bounds, and Reproducible Simulations

Parham Ghayour

Sorbonne Université (Paris VI), Paris, France

19 October 2026

Abstract

Magnitude comparison in a residue number system requires recovering order information that is absent from individual residues. We develop a corrected analysis of the author’s three-modulus cluster construction and extend it into a workload-aware study of exact comparison. For arbitrary pairwise-coprime moduli, the cluster and subgroup indices are precisely mixed-radix digits; modular inversion gives a closed-form encoder and a direct lexicographic decision rule. Beyond this representation result, we derive a sharp finite-cardinality lower bound on unresolved comparisons for ordered tags, an exact prefix-collision formula for digit-comparison work, an optimal modulus order under independent uniform inputs, and an exact cyclic-shift collision law for correlated inputs. For nonuniform independent operands, a dynamic program computes the optimal contiguous tag partition under a fixed tag-count budget. Executed exact-integer checks cover 364,242 reconstruction inputs, 380,299 exhaustive operand pairs, and 60,384 large-number and boundary pairs, with zero discrepancies. Seven figures report structure, decision regions, workload effects, partition optimization, and measured software timings. In a specified synthetic skewed workload, three optimized tags reduce the theoretical strict-tie probability from 0.85596 to 0.29192. This is a reduction in fallback frequency, not a measured end-to-end speedup. Python microbenchmarks show that direct comparison improves on the eager predecessor, while a simple CRT baseline has a lower median time at every tested width. The contribution is a rigorous refinement and reproducible workload analysis, not a new number representation or a claim of universal hardware superiority.

Keywords: residue number system; mixed-radix conversion; exact comparison; ordered partitions; workload analysis; reproducible computation.

1 Introduction and contribution boundaries

An RNS represents an integer through its residues modulo pairwise-coprime integers. Modular arithmetic is channel-local, but order is not: neither a single residue nor componentwise residue ordering generally determines the order of canonical integers. Classical mixed-radix conversion (MRC) and Chinese remainder theorem (CRT) reconstruction recover sufficient positional information to compare values [1, 2].

The author’s 2016 preprint proposed assigning numbers to ordered clusters and using the cluster of a modular difference to resolve ties [4]. Two subsequent manuscripts clarified that the extracted cluster/subgroup coordinates are mixed-radix digits. The present article retains that correction, replaces inherited experimental statements by a fresh executable study, and extends the analysis from correctness to the distribution of comparison work.

The central research question is therefore narrower and more testable than whether clustering is a new numeral system: *given an exact mixed-radix comparator, how do tag capacity, modulus*

ordering, and the operand distribution control the frequency and cost of unresolved decisions? A useful answer must distinguish the mathematical decision stage from the cost of extracting its inputs.

The contributions developed here are:

1. An assumption-complete derivation of the three-channel encoder, its equivalence to MRC, and the original and direct comparison rules.
2. A sharp collision bound for finite tags, together with its equality conditions and its interpretation as an optimal ordered partition under uniform inputs.
3. Exact formulas for expected digit probes and deferred encodings; a descending-radix optimality theorem restricted to the uniform decision-stage model; and an exact correlated-input shift law.
4. A dynamic-programming formulation for optimal contiguous tags under a known nonuniform independent workload, with an independent finite verification of the computed three-tag optimum.
5. Reproducible arithmetic checks, workload simulations, and software comparisons that include unfavorable as well as favorable results.

These are additions to the supplied corrected drafts. MRC, CRT, collision minimization, and dynamic programming are established ideas; no priority claim is made for their general principles. The value of the present formulation is their explicit connection to this comparator, complete proofs, and quantitative verification. A comprehensive priority analysis of all RNS comparison literature is not claimed.

2 Exact representation and extraction

2.1 Input contract

Let $p_1, p_2, p_3 \geq 2$ be pairwise coprime, and set

$$M = p_1 p_2 p_3, \quad L = p_2 p_3, \quad x_j = [X]_{p_j}, \quad 0 \leq X < M.$$

Here $[a]_m = a - m[a/m]$ is always the least nonnegative residue, including for negative a . The order of the moduli matters, but primality and increasing modulus order are unnecessary. Without the canonical-range assumption, the residues identify only a congruence class, and unrestricted integer order is undefined.

Partition the range into consecutive clusters $C_k = [kL, (k+1)L) \cap \mathbb{Z}$, $0 \leq k < p_1$. Precompute

$$\gamma = [p_3^{-1}]_{p_2}, \quad \alpha = [L^{-1}]_{p_1}, \quad \beta = [\alpha p_3]_{p_1}. \quad (1)$$

All inverses exist by pairwise coprimality.

Theorem 2.1 (Cluster digits). *The formulas*

$$i(X) = [\gamma(x_2 - x_3)]_{p_2}, \quad (2)$$

$$k(X) = [\alpha(x_1 - x_3) - \beta i(X)]_{p_1} \quad (3)$$

recover the unique decomposition

$$X = Lk(X) + p_3 i(X) + x_3, \quad 0 \leq k < p_1, \quad 0 \leq i < p_2. \quad (4)$$

In particular, $k = \lfloor X/L \rfloor$ and $i = \lfloor (X \bmod L)/p_3 \rfloor$.

Proof. Euclidean division gives $X = Lk + u$ with $0 \leq u < L$. Since p_3 divides L , $u \equiv x_3 \pmod{p_3}$, hence $u = x_3 + p_3i$ for a unique $0 \leq i < p_2$. Reducing this identity modulo p_2 and multiplying by p_3^{-1} gives the formula for i . Reducing $X = Lk + x_3 + p_3i$ modulo p_1 gives the formula for k . Its canonical solution is the true quotient because that quotient lies in $[0, p_1)$. This proves both existence and uniqueness. $\square$

The predecessor's subgroup map $S(r, i) = [r - ip_3]_{p_2}$ is a permutation in i for each r . Solving $S(x_2, i) = [x_3]_{p_2}$ yields the same i . Arithmetic inversion thus replaces subgroup search. A wraparound induction that assumes quotient increments are always zero or one is unnecessary and can fail when the moduli are reordered; Euclidean division has no such restriction.

2.2 Equivalence to MRC and an alternative arithmetic schedule

Define $K(X) = (k(X), i(X), x_3)$, from most to least significant. These are exactly the MRC digits for least-to-most radix order (p_3, p_2, p_1) . An ordinary MRC schedule computes

$$i = \left[(x_2 - x_3)p_3^{-1} \right]_{p_2}, \quad h = \left[(x_1 - x_3)p_3^{-1} \right]_{p_1}, \quad k = \left[(h - i)p_2^{-1} \right]_{p_1}.$$

The expanded cluster formula eliminates h algebraically; it does not introduce a distinct representation.

A factored alternative is

$$k = [\alpha(x_1 - x_3 - p_3i)]_{p_1}. \quad (5)$$

This schedule has two explicit final remainder operations per operand, but also forms p_3i and a wider intermediate product with α . Counting remainder symbols alone would misstate its cost. Both schedules are benchmarked separately.

For $(p_1, p_2, p_3) = (3, 5, 7)$, $(\gamma, \alpha, \beta) = (3, 2, 2)$:

X	(x_1, x_2, x_3)	k	i	x_3
11	(2, 1, 4)	0	1	4
20	(2, 0, 6)	0	2	6
52	(1, 2, 3)	1	2	3
63	(0, 3, 0)	1	4	0
96	(0, 1, 5)	2	3	5

In particular, (0, 1, 5) represents 96, not 52. Figure 1 contrasts the oscillating raw residues with the extracted hierarchy.

3 Exact unsigned and signed comparison

Theorem 3.1 (Direct comparison). *For canonical X, Y , $X > Y$ if and only if $K(X) >_{\text{lex}} K(Y)$. Equality holds exactly when all three residues, or equivalently all three key digits, agree.*

Proof. In (4), $0 \leq p_3i + x_3 < L$, so a nonzero cluster difference determines the order. With equal clusters, $0 \leq x_3 < p_3$, so a nonzero subgroup difference determines the order. With both digits equal, the remaining order is that of x_3 . Uniqueness proves the equality statement. $\square$

Theorem 3.2 (Original difference rule). *Let $Z = [X - Y]_M$. If $X \neq Y$ and $k(X) = k(Y)$, then $X > Y$ exactly when $k(Z) = 0$, and $X < Y$ exactly when $k(Z) = p_1 - 1$. If the cluster digits differ, their order already determines the answer.*

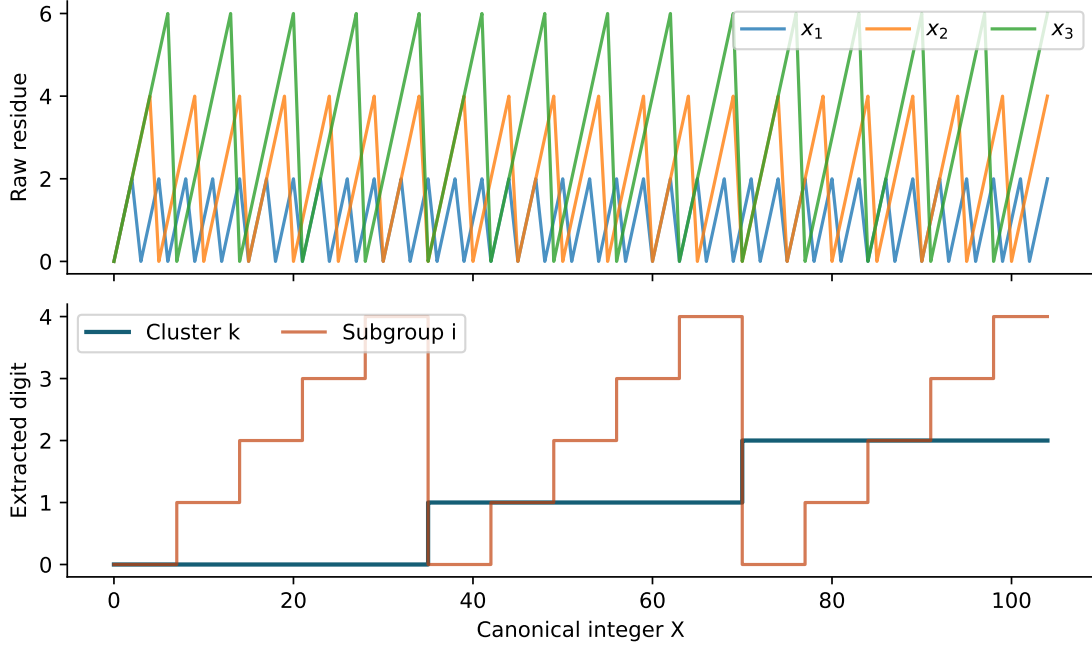


Figure 1: Exact enumeration for $(3, 5, 7)$, $M = 105$. The cluster digit is monotone; subgroup and residue digits reset at their respective boundaries. The complete lexicographic key, not an individual low-order digit, preserves order.

Proof. For a strict same-cluster pair, $1 \leq |X - Y| \leq L - 1$. A positive difference has $Z \in [1, L - 1]$, while a negative difference has $Z \in [M - L + 1, M - 1]$. These lie in the first and last clusters, respectively. For different clusters, Theorem 3.1 applies. $\square$

The equality condition is essential because $X = Y$ also yields $k(Z) = 0$. The first/last-cluster characterization is asserted only for same-cluster operands. For arbitrary pairs, a modular difference alone does not retain the sign of the ordinary difference.

3.1 Signed convention

Let $H = \lceil M/2 \rceil$ and $\sigma(X) = X - M\mathbf{1}_{X \geq H}$. The signed range is $[-\lfloor M/2 \rfloor, \lceil M/2 \rceil - 1]$. Set $s(X) = \mathbf{1}_{X \geq H}$ by comparing $K(X)$ with the constant $K(H)$. Then the signed order is the lexicographic order of

$$K_s(X) = (1 - s(X), k(X), i(X), x_3). \quad (6)$$

Indeed, negative values precede nonnegative values, and within each sign interval σ is a translation and therefore increasing. For even p_1 , the sign threshold is simply $k \geq p_1/2$. This extension does not provide overflow detection or a signed arithmetic specification.

3.2 Magnitude intervals

The cluster difference also supplies a certified interval:

$$(k_X - k_Y)L - (L - 1) \leq X - Y \leq (k_X - k_Y)L + (L - 1). \quad (7)$$

Both endpoints are attainable for fixed cluster indices by selecting opposite lower-offset extremes. Consequently, a cluster-level decision can certify that a difference exceeds a chosen threshold when this entire interval lies above it. Otherwise the finer digits must be consulted. This is an order certificate, not evidence that the cluster can be computed without the lower digits.

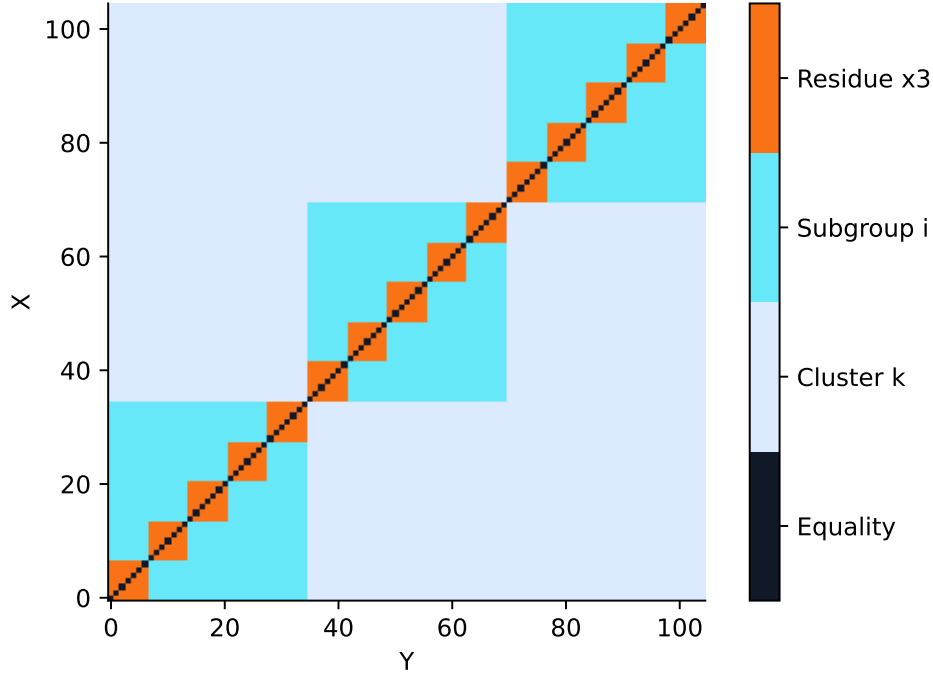


Figure 2: All 11,025 ordered pairs for (3, 5, 7). Colors identify the first differing key digit; the diagonal is equality. The nested blocks visualize exactly where a cluster-only decision is insufficient. This is an exhaustive computed decision map.

4 Sharp bounds for ordered tags

A complete per-operand tag that resolves all comparisons must be injective: if $f(x) = f(y)$ for $x \neq y$, the tag pairs for (x, x) and (x, y) coincide but require different outputs. Such a tag needs at least M values and $\lceil \log_2 M \rceil$ fixed-width bits.

A smaller tag can still resolve many pairs if exact fallback remains available. Let f map the M canonical integers to q nonempty tags. In an *ordered-tag comparator*, different tags settle order and equal tags trigger refinement unless equality has already been detected from the full residues. Such tags must be monotone: their fibers are contiguous intervals. The following lower bound itself does not need monotonicity.

Theorem 4.1 (Finite tag-capacity bound). *For independent uniform X, Y and fiber sizes $n_1, \dots, n_q$, the probability of a strict tag tie is*

$$P_{\text{strict}} = \frac{\sum_{j=1}^q n_j^2 - M}{M^2}. \quad (8)$$

Write $M = aq + r$, $0 \leq r < q$. Then

$$P_{\text{strict}} \geq \frac{(q-r)a^2 + r(a+1)^2 - M}{M^2} \geq \frac{1}{q} - \frac{1}{M}. \quad (9)$$

The first bound is attained by balanced contiguous intervals with sizes a or $a+1$. The second bound is attained when q divides M and all sizes equal M/q .

Proof. There are n_j^2 ordered pairs in fiber j , of which n_j are equal. Summing gives (8). If two fiber sizes differ by at least two, moving one element from the larger to the smaller decreases their sum of squares by $2(n_{\text{large}} - n_{\text{small}} - 1) > 0$. A minimizer therefore has sizes differing by at most one, giving the first bound. Cauchy–Schwarz gives $\sum n_j^2 \geq M^2/q$, with equality exactly for equal sizes. Balanced intervals realize the minimizing sizes while preserving tag order. $\square$

For cluster tags, $q = p_1$ divides M and every fiber has size L . Thus

$$P_{\text{strict}} = \frac{1}{p_1} - \frac{1}{M}. \quad (10)$$

Equal clusters are optimal among p_1 -tag schemes *for this uniform strict-tie objective*. This statement neither minimizes the arithmetic needed to generate the tags nor supplies a lower bound for every RNS comparator. The comparison interface still contains the original residues or another exact refinement representation.

If a system stores only a q -valued tag and discards all remaining information, exact fallback is impossible unless $q \geq M$. The lower bound is therefore a storage–ambiguity tradeoff for an additional summary, not a compression theorem for full exact order.

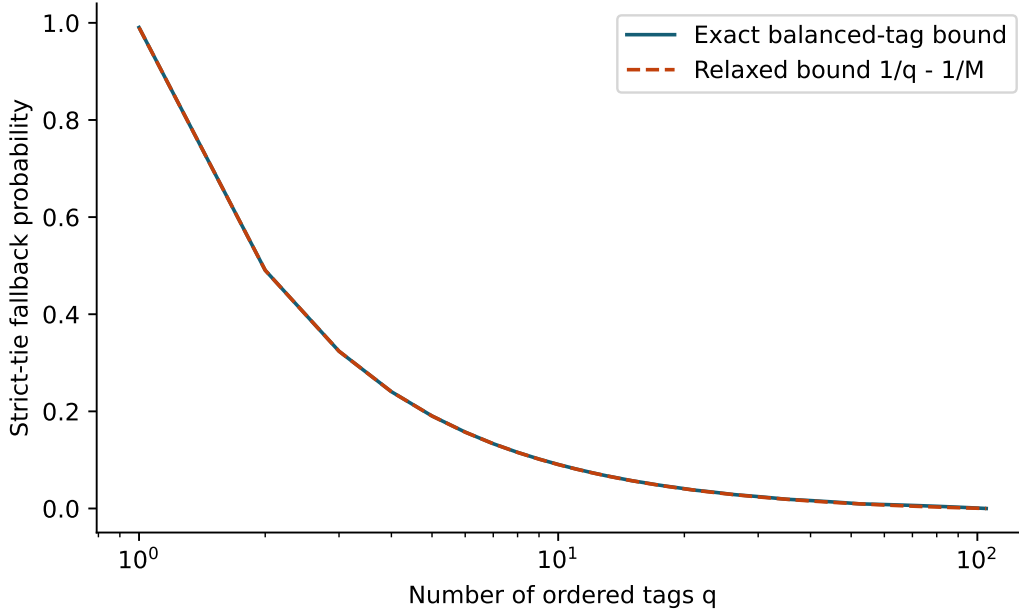


Figure 3: Sharp integer-partition lower bound for $M = 105$, together with the relaxed bound. Coincidence occurs at divisors of M . At $q = M$, each tag identifies one value and no strict ties remain.

5 Workload laws and modulus ordering

5.1 Prefix collisions and expected decision work

Consider n mixed-radix digits with radices $r_1, \dots, r_n$ in *most-to-least* significance order. Let A_j denote equality of the first j digits. A digit comparator inspects the next digit exactly when all previous digits agree. Counting one probe per compared digit, including the last digit on equal inputs, gives the pointwise identity

$$D = 1 + \sum_{j=1}^{n-1} \mathbf{1}_{A_j}, \quad \mathbb{E}[D] = 1 + \sum_{j=1}^{n-1} \mathbb{P}(A_j). \quad (11)$$

This holds for any joint operand distribution, including correlated pairs. It describes digit probes after keys exist, not encoder operations or hardware cycles. An implementation that first detects equality from residues has a different probe schedule and must account for that test separately.

Under independent uniform inputs, each prefix has $r_1 \cdots r_j$ equally populated classes, so

$$\mathbb{E}[D] = 1 + \sum_{j=1}^{n-1} \frac{1}{r_1 \cdots r_j}. \quad (12)$$

In the three-channel convention this is $1 + 1/p_1 + 1/(p_1 p_2)$.

Theorem 5.1 (Uniform optimal order). *For a fixed set of radices and equal unit digit-probe costs, descending radix order from most to least significant minimizes $\mathbb{E}[D]$ under independent uniform operands.*

Proof. Suppose adjacent radices $a < b$ occur after a prefix product P . Swapping them changes only the term corresponding to the prefix ending at the first of those positions: $1/(Pa)$ is replaced by $1/(Pb)$. Earlier terms are unchanged and later prefix products contain both factors. The cost decreases. Repeatedly swapping inversions yields descending order. $\square$

For $(3, 5, 7)$, order $(7, 5, 3)$ gives $1 + 1/7 + 1/35 = 1.17143$ probes, while $(3, 5, 7)$ gives 1.4. Their encoder costs need not be equal. Wider top digits, different constants, input correlations, and implementation-specific arithmetic can reverse an end-to-end ranking.

5.2 Deferred difference encoding

Define a schedule that always forms the two operand encodings, resolves equality from canonical residues, and encodes Z only for a strict cluster tie. Its encoding count N_E obeys

$$N_E = 2 + \mathbf{1}_{\{k_X = k_Y, X \neq Y\}}, \quad \mathbb{E}[N_E] = 2 + \mathbb{P}(k_X = k_Y, X \neq Y). \quad (13)$$

The eager predecessor uses three encodings and the direct rule uses two. Under uniform independence, (10) gives $2 + 1/p_1 - 1/M$ for the deferred schedule. Equality testing before either encoder would change all three schedules on equal pairs; that variant is not the schedule counted here.

5.3 Correlated operands: an exact displacement law

Theorem 5.2 (Cyclic-shift cluster collision). *Let X be uniform on $[0, M)$, let $Y = [X + \delta]_M$, and let $0 \leq \delta < M$. For equal-width clusters of length $L = M/p_1$,*

$$\mathbb{P}(k_X = k_Y) = \frac{(L - \delta)_+ + (L - (M - \delta))_+}{L}, \quad t_+ = \max(t, 0). \quad (14)$$

For $\delta \neq 0$, this is also the strict-tie probability; for $\delta = 0$ the strict-tie probability is zero.

Proof. Within any one cluster, nonwrapping shifts contribute $(L - \delta)_+$ starting positions. Shifts wrapping around the full range contribute $(L - (M - \delta))_+$ positions. These cases are disjoint; multiplying their sum by the p_1 clusters and dividing by $M = p_1 L$ gives the formula. For a nonzero canonical shift, $X \neq Y$ for every X . $\square$

For adjacent pairs ($\delta = 1$), the strict-tie rate is $1 - 1/L$, which can be near one even when the uniform independent estimate $1/p_1 - 1/M$ is small. Thus two workloads on the same modulus set can induce very different deferred costs.

A concrete counterexample to distribution-free modulus-order optimality is the fixed pair $(20, 21)$ in range $M = 105$. For $p_1 = 5$, $L = 21$ and the pair crosses a cluster boundary, so no fallback is needed. For the larger $p_1 = 7$, $L = 15$ and both operands lie in the same cluster. This does not contradict Theorem 5.1, whose independence and uniformity assumptions are essential.

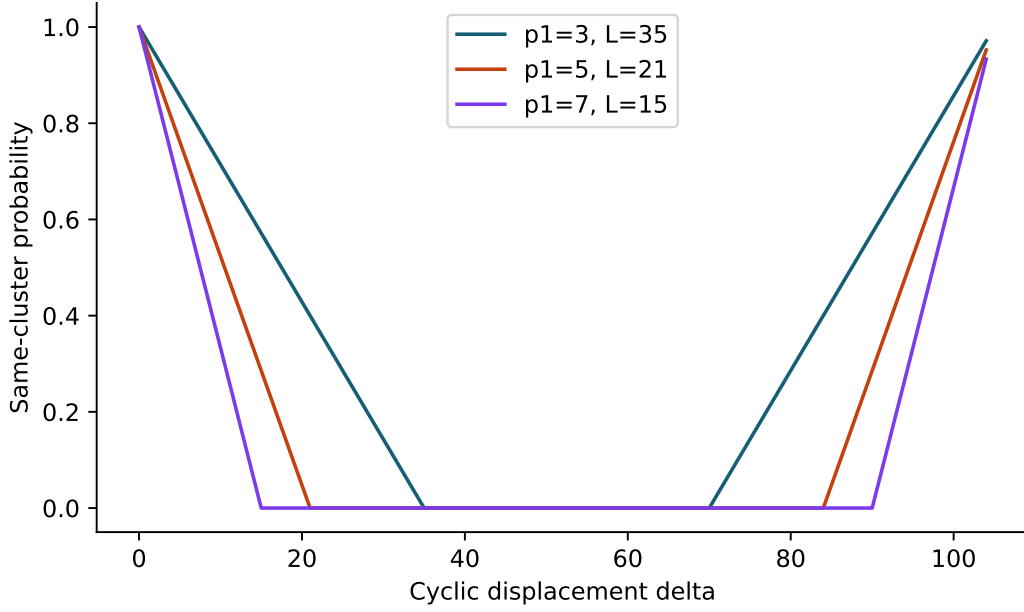


Figure 4: Exact cyclic-shift collision curves for $M = 105$, obtained by enumerating all starting values at each displacement. All 630 order/displacement cases agree with Theorem 5.2. The point $\delta = 0$ is equality, not a strict fallback.

6 Nonuniform inputs and optimized contiguous tags

The preceding results suggest adapting the partition when the input distribution is known. This section optimizes a probabilistic fallback objective, without assuming that an arbitrary partition is cheap to extract directly from residues.

Let X, Y be independent with a common mass function w_x over $x = 0, \dots, M-1$, and let tag intervals have probability masses $\pi_1, \dots, \pi_q$. Then

$$P_{\text{strict}} = \sum_{j=1}^q \pi_j^2 - \sum_{x=0}^{M-1} w_x^2. \quad (15)$$

The subtracted equality probability is independent of the partition. Hence minimizing fallback is equivalent to minimizing the squared interval masses. For a general correlated joint law, the corresponding formula instead sums joint probability inside same-tag squares and subtracts $\mathbb{P}(X = Y)$; marginal masses alone do not determine the result.

Proposition 6.1 (Optimal partition recurrence). *Write $W(j) = \sum_{x=0}^{j-1} w_x$ with $W(0) = 0$. The minimum sum of squared masses for a partition of $\{0, \dots, j-1\}$ into s nonempty consecutive intervals satisfies*

$$F(s, j) = \min_{s-1 \leq t < j} \left\{ F(s-1, t) + (W(j) - W(t))^2 \right\}, \quad (16)$$

with $F(0, 0) = 0$, $F(0, j) = +\infty$ for $j > 0$, and infeasible states set to $+\infty$. Its optimal strict-tie probability is $F(q, M) - \sum_x w_x^2$.

Proof. In any feasible partition, the last interval is $[t, j)$ for some $s-1 \leq t < j$, with squared probability mass $(W(j) - W(t))^2$. The preceding $s-1$ intervals optimally partition $[0, t)$; otherwise replacing them would improve the full solution. Minimizing over all last cuts gives the recurrence. Backtracking minimizing cuts constructs a globally optimal partition. $\square$

The straightforward implementation uses $O(qM^2)$ arithmetic work and $O(qM)$ storage including cut reconstruction. It is therefore intended for finite manageable domains or as an offline design study, not as an efficient optimizer over cryptographic-size dynamic ranges. Zero-probability regions can produce multiple optimal cuts. Exact probability inputs admit an exact-arithmetic implementation; the experiment below uses double-precision probabilities and independently checks the three-tag optimum to a 10^{-12} objective tolerance.

6.1 Synthetic skewed-distribution experiment

Use $M = 105$ and $w_x = \exp(-x/12) / \sum_{u=0}^{104} \exp(-u/12)$. With three tags, equal-width cuts at 35 and 70 give a theoretical strict-tie probability of 0.855957. The optimized cuts at 5 and 13 give 0.291915, a relative reduction of approximately 65.9%. All 5,356 possible pairs of interior cuts were separately enumerated to verify this three-tag optimum.

An independent seeded sample of 100,000 pairs gives observed rates 0.85452 and 0.29409, respectively. Their approximate Monte Carlo standard errors are 0.00112 and 0.00144. These samples test the formula under the specified distribution; they do not constitute evidence of performance on an application workload. Figure 5 shows the distribution, cuts, and the objective for 1–15 tags.

Implementation implication. An arbitrary optimized tag can be extracted by comparing the full key against stored boundary keys. If key extraction is needed afresh for every pair, this extra partitioning may offer no benefit over simply comparing the keys. Its plausible use is as a cached summary in a repeated-comparison workload with exact fallback retained. Cache maintenance, invalidation after RNS arithmetic, and boundary-search cost must be charged. The measured reduction is solely in fallback frequency; no end-to-end acceleration is asserted.

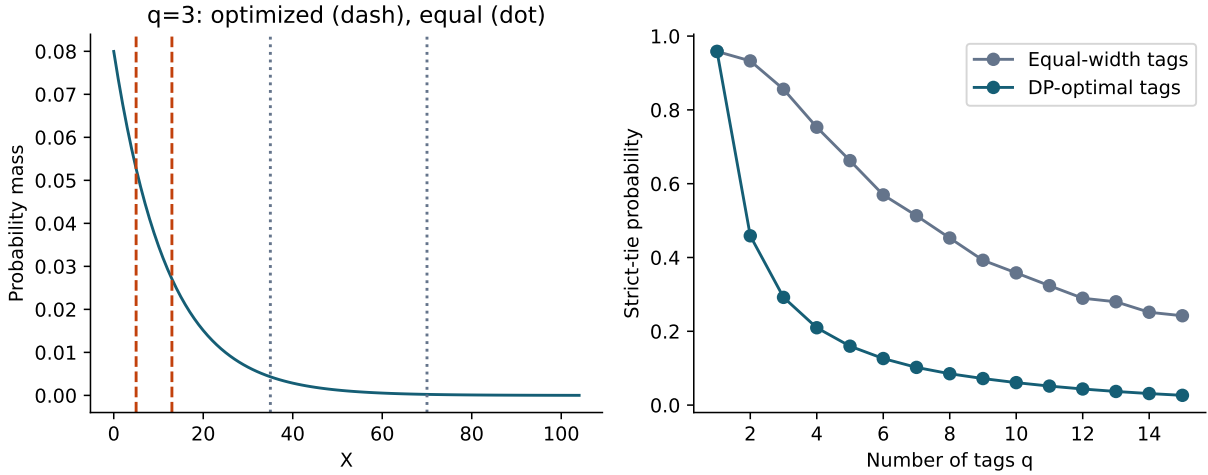


Figure 5: Distribution-aware partitioning. Left: the synthetic probability mass and the optimized/equal-width three-tag cuts. Right: theoretical strict-tie probabilities from the dynamic program and equal-width integer partitions. The improvement concerns an explicitly stated iid distribution and objective.

7 Complexity, storage, and implementation contracts

7.1 Arithmetic and bit costs

With precomputed constants, the expanded three-channel encoder uses three constant multiplications and two final modular reductions as written, together with three subtractions. If each product is reduced separately, this becomes three modular constant products and modular subtractions. These are alternative arithmetic schedules, not interchangeable physical cost counts. Each operand also carries cross-channel values that may need normalization in a fixed-width implementation.

For channel widths at most b , all intermediates in the three-channel formulas have $O(b)$ bits (with different constant factors), and elementary exact arithmetic gives an $O(b^2)$ upper bound per encoder. Thus a fixed operation count for three channels is not constant bit complexity. The simple n -channel MRC recurrence uses $n(n-1)/2$ modular constant products; its conservative schoolbook bound is $O(n^2b^2)$. Parallel conversion algorithms require their own dependency and resource analysis [3, 6].

Inversion is preprocessing. A full subgroup table uses p_2^2 entries, whereas the map $d \mapsto [\gamma d]_{p_2}$ needs only p_2 entries. Both grow exponentially in the bit width of p_2 ; arithmetic avoids this particular table growth.

7.2 Decision trees and cached keys

Concatenating canonical binary digit fields preserves lexicographic order, although the concatenation generally differs from the ordinary binary representation of X . Its width $B = \sum_j \lceil \log_2 p_j \rceil$ is less than $\log_2 M + 3$ for three channels. For $(2, 3, 5)$, the key has six bits although the represented range $0, \dots, 29$ fits in five bits. The gaps do not affect order.

A bit segment can be summarized by equality and greater-than flags (E, G) . Combining a more-significant segment H with a less-significant segment L gives

$$(E_H, G_H) \circ (E_L, G_L) = (E_H E_L, G_H \vee E_H G_L).$$

Both parenthesizations of three segments yield $E_H E_M E_L$ and $G_H \vee E_H G_M \vee E_H E_M G_L$, proving associativity. A balanced tree therefore has $O(B)$ cells and $O(\log B)$ combination depth, excluding key generation. This is standard comparison logic, not a hardware speed result.

If N values are compared repeatedly, converting once and caching their keys can move conversion cost from each comparison to each value update. CRT reconstruction can be cached in the same way. A fair application comparison must offer equivalent caching opportunities to both baselines. Neither technique permits stale keys after modular updates.

7.3 Invalid values and faults

The mathematical interface accepts only canonical residues, with coprime moduli. Software must reject noncoprime bases or explicitly normalize noncanonical residues under a stated interface. Our tests construct canonical inputs and do not test a production input-validation layer. Every canonical tuple corresponds to some integer by CRT, so range-valid residues do not provide fault detection: a changed but still canonical residue can silently represent another number. Redundant channels or a separate integrity mechanism would be needed for that purpose.

8 Executed study and results

8.1 Reproducibility protocol

All results in this article were generated for this revision, rather than copied from the earlier drafts. The accompanying source bundle contains `experiments.py`, `partition_experiment.py`, `results.json`, `partition_results.json`, a timing CSV, figures, and the $\text{\LaTeX}$ source. The reference arithmetic uses Python integers; correctness is not subject to fixed-width overflow. The main random seed is 20261019; the partition Monte Carlo seed is 20261020.

Check	Cases	Discrepancies
Digit identities over 612 ordered coprime triples	364,242	0
Direct, eager/deferred difference, and signed rules	380,299 pairs	0
Large-number random and selected boundary pairs	60,384 pairs	0
General MRC reconstruction, two to five channels	4,731 inputs	0
Cyclic-shift law over six modulus orders	630 cases	0
Contiguous-partition lower bound, $2 \leq M \leq 12$	4,094 partitions	0
Independent cut enumeration for skewed $q = 3$ optimum	5,356 cuts	0

The rows represent different validation tasks and should not be added into a single homogeneous “test count.” The pair checks validate several rules on each pair. The last row uses a floating-point tolerance, while the preceding correctness and combinatorial checks use exact integers.

The reconstruction sweep uses every ordered pairwise-coprime triple from $\{2, \dots, 15\}$ and every canonical value in its range. This includes composite moduli and all permutations. The exhaustive pair bases are $(2, 3, 5)$, $(3, 5, 7)$, $(5, 7, 11)$, $(4, 9, 5)$, $(7, 3, 8)$, $(11, 5, 3)$, $(2, 5, 3)$, $(3, 2, 5)$, $(5, 3, 2)$, and $(8, 9, 5)$. Their full pair counts also verify the predicted uniform collision and digit-probe totals.

The large-number tests use $(2^b - 1, 2^b, 2^b + 1)$ for $b \in \{8, 16, 32, 64, 128, 256\}$, with 10,000 random pairs and 64 selected boundary pairs per width. The largest dynamic range has 768 bits. These special-family tests extend arithmetic coverage; they do not establish correctness on every large base. Universal correctness follows from the proofs. The generalized reconstruction tests use $(2, 3)$, $(3, 5, 7, 11)$, $(4, 9, 5, 7)$, and $(2, 3, 5, 7, 11)$.

8.2 Workload simulations

For $M = 105$, each of six orders of $(3, 5, 7)$ is evaluated on the same 100,000 pairs per workload:

- **Uniform:** independent uniform operands.
- **Adjacent:** uniform X and $Y = (X + 1) \bmod 105$.
- **Equal-heavy:** uniform X ; with probability 0.8 set $Y = X$, otherwise draw Y uniformly and independently.
- **Boundary pair:** always $(X, Y) = (20, 21)$, a deterministic stress case.

The first two measure independent and locally correlated behavior. Equal-heavy inputs separate the equality path from strict fallback. In that mixture, strict-tie probability is $0.2(1/p_1 - 1/M)$; same-cluster probability without excluding equality would be much larger. The fixed pair demonstrates why a uniform-optimal order need not be optimal for a particular workload.

Figure 6 reports strict-fallback rates and digit probes. Error bars on fallback rates are $1.96\sqrt{\hat{p}(1 - \hat{p})/N}$; they describe sampling variability only, not a simultaneous confidence guarantee across all curves. The deterministic pair has no sampling uncertainty. The digit-probe graph uses the raw lexicographic schedule of (11), while deferred encoding counts use equality exclusion in (13).

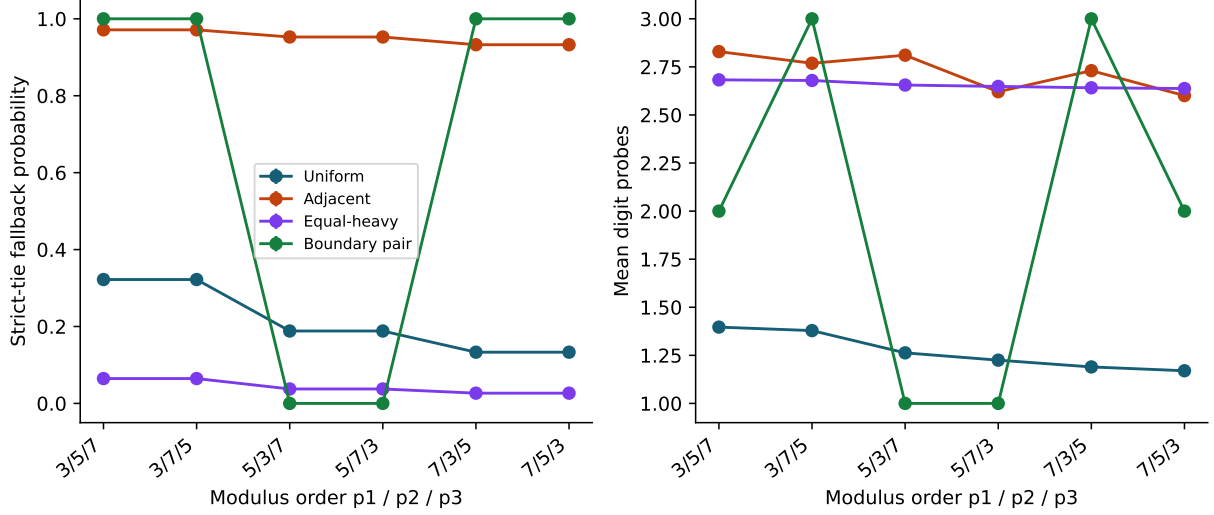


Figure 6: Measured workload dependence across all six modulus orders. Left: the probability of requiring the predecessor’s third encoding under the stated deferred schedule. Right: digit probes after key generation. Largest-first is optimal for the uniform probe objective, but the fixed boundary pair favors $p_1 = 5$ over $p_1 = 7$.

8.3 Software baseline definitions

The timing study compares six exact implementations:

1. Expanded direct key extraction, followed by tuple comparison.
2. Factored direct extraction using (5).
3. Unrolled three-channel MRC in the same digit order.
4. Unrolled CRT reconstruction, followed by integer comparison.
5. Eager predecessor: two operand encodings plus the difference encoding on every pair.
6. Deferred predecessor: two operand encodings plus the difference encoding only on a strict cluster tie.

The CRT implementation precomputes coefficients and reconstructs the canonical integer as follows:

$$c_j = (M/p_j) \left[(M/p_j)^{-1} \right]_{p_j}, \quad X = \left[\sum_j c_j x_j \right]_M.$$

The MRC baseline is explicitly unrolled to avoid charging a generic nested-loop framework only to that method. The direct functions still have different Python call and unpacking overhead from closure-based baselines. The measured ranking is therefore that of these source implementations in this runtime, not a pure arithmetic-operation or optimized native-code ranking.

For each width in $\{8, 16, 32, 64, 128\}$, a seeded arbitrary pairwise-coprime triple is selected in $[2^{b-1}, 2^b)$. Moduli need not be prime. There are 6,000 independent uniform pairs per width and nine timing rounds, with randomized method order each round. Constants, input residues, and correctness checks are outside the timed region; key generation and decision are inside it. Each method is checked against the known integer order for every benchmark pair before timing. The timed batch includes a common Python loop and checksum accumulation. All raw batch observations and exact triples are saved.

The run used Python 3.12.14 on Linux x86_64, with an AMD EPYC 9V74 host CPU. The execution environment is shared: CPU pinning, frequency control, isolation, and native-code optimization were not applied. The nine rounds are repeat measurements on one data batch, not nine independent machines or application datasets. Medians and interquartile bands are reported without claims of statistical superiority.

Table 1: Median microseconds per comparison. Each entry is the median of nine 6,000-pair batches; conversion is included and setup is excluded.

Method	8 bits	16 bits	32 bits	64 bits	128 bits
Direct expanded	0.584	0.611	1.007	1.199	1.375
Direct factored	0.530	0.656	1.056	1.239	1.583
MRC unrolled	0.544	0.626	1.082	1.358	1.690
CRT unrolled	0.421	0.587	0.653	0.819	1.370
Predecessor eager	1.376	1.553	2.198	2.484	2.781
Predecessor deferred	0.611	0.695	1.040	1.295	1.428

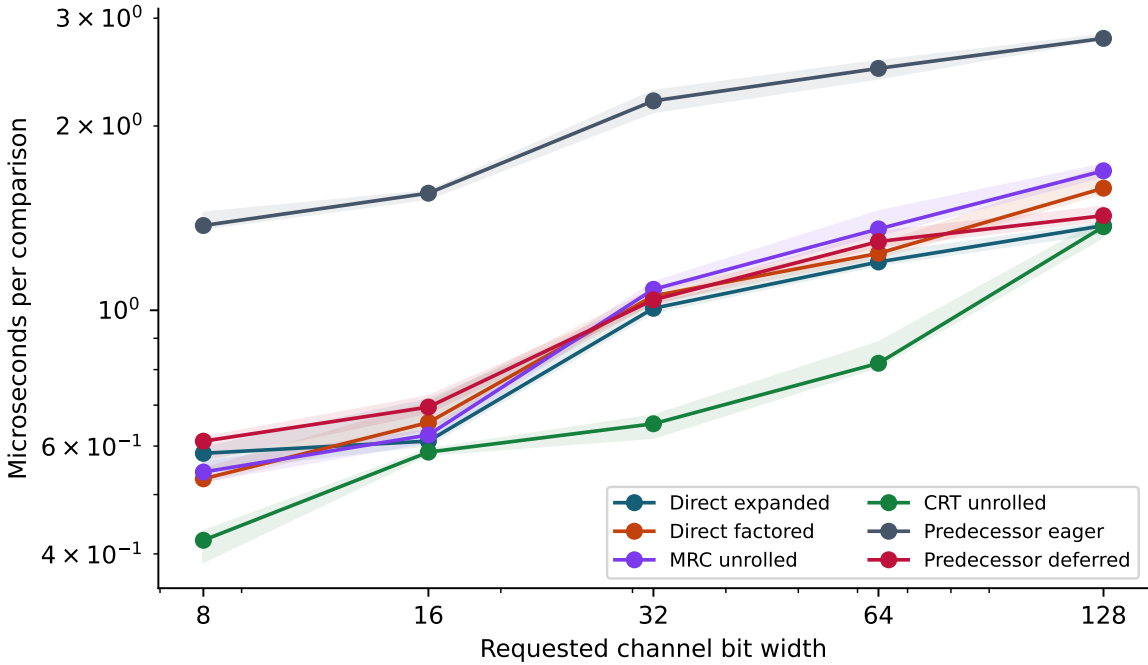


Figure 7: Measured Python time per comparison, including conversion, on arbitrary coprime triples. Lines show nine-round medians and bands show the interquartile range. Both axes use logarithmic scaling. This is a software microbenchmark, not gate-level or physical hardware timing.

8.4 Interpretation of the results

The direct formulation eliminates the eager predecessor’s unnecessary third encoding and improves measured time in this implementation. The deferred predecessor is already close to the direct path for uniform inputs, because same-cluster ties are uncommon for large p_1 . Counting a universal one-third speed improvement from encoder counts would therefore be incorrect.

CRT has a lower median time than the expanded direct implementation at every tested width in this run. At 128 bits the medians are nearly equal (1.370 versus 1.375 microseconds); this small difference does not support a claim of a stable speed advantage. Standard unrolled MRC

remains competitive, as expected from the algebraic equivalence. The factored expression does not consistently improve time despite having fewer explicit final reductions; wider intermediate arithmetic and interpreter effects matter. These observations argue for benchmarking the full implementation rather than selecting a method by one operation count.

The partition experiment establishes a different result: fewer unresolved pairs can be obtained by adapting interval cuts to a known skewed distribution. It does not time a cached partition architecture, so it must not be conflated with the six-method benchmark. Likewise, digit-probe reductions omit key extraction. The experiments identify where an application-level study could be worthwhile, while preserving those distinctions.

9 Relation to existing work and remaining limitations

MRC is classical [1, 2]; parallel conversion has a long history [3], and CRT-based parallel reverse conversion remains an active design topic [6]. The present key is a three-digit MRC representation, and its general recurrence is not a new conversion algorithm. The original cluster comparator [4] motivates the particular formulas and difference-rule correction.

Modified diagonal-function comparators offer a different order mapping and have been evaluated using synthesized hardware [5]. We do not reproduce those circuits or compare their published technology numbers with Python runtimes. Didier et al. describe a comparison approach using an available additional modulus and one MRC-based extension [7]. That redundant input changes the resource contract; our benchmark has only the three original residue channels. Luguterah et al. report a sign-detection-based RNS comparison design [8]; it is a relevant recent reference, not an implemented baseline here. These sources provide context, not a complete state-of-the-art performance comparison.

The main limitations are concrete. First, workload laws describe decision work or fallback frequency, not the cost of obtaining order tags. Second, the DP optimizes a known iid distribution and an interval-mass objective; it has quadratic dependence on the full domain size and has not been evaluated under distribution drift. Third, the benchmark has one arbitrary triple per width, three channels, one runtime, and no production application. Fourth, no HDL simulator, synthesis tool, post-route analysis, or physical device measurement was used in this revision. A Python exact-integer simulation is not an FPGA experiment. Finally, finite tests cannot establish novelty or replace the proofs.

An informative next study would implement the direct, MRC, CRT, and appropriate diagonal-function baselines in a common native or HDL toolchain. It should hold the modulus set, validity interface, cached-state assumptions, and arithmetic widths fixed; separate preprocessing from conversion and comparison; and evaluate real operand traces as well as synthetic ones. A cached adaptive-tag design should include boundary lookup and update costs, and train its partitions on a separate trace before measuring held-out comparisons. These are proposed experiments, not completed results.

10 Conclusion

The cluster construction is an exact mixed-radix comparator with a direct proof for arbitrary pairwise-coprime triples. Its original modular-difference rule is correct under an explicit equality and same-cluster contract, while direct key comparison removes the third encoding. The extended analysis provides a sharp tag-capacity bound, uniform-order optimality for digit probes, an exact correlated-shift law, and an optimal finite-domain partition recurrence for a known iid distribution. Reproducible computations verify the identities and expose substantial workload dependence. The software measurements also show the limit of the structural simplification: CRT has the lowest median times in this tested Python setting, with a near tie at the largest width. The defensible outcome is a mathematically complete, experimentally transparent refinement with a precise path toward application and hardware evaluation.

Revision assistance. This revision was prepared with AI assistance for derivations, reference checking, programming, experimental execution, visualization, and manuscript drafting. The author should review the proofs, source code, affiliation, and references before submission. The manuscript and computations have not undergone external peer review.

References

- [1] H. L. Garner, “The residue number system,” *IRE Transactions on Electronic Computers*, EC-8(2), 140–147, 1959. doi:10.1109/TEC.1959.5219515.
- [2] N. S. Szabo and R. I. Tanaka, *Residue Arithmetic and Its Applications to Computer Technology*. McGraw-Hill, 1967.
- [3] C.-H. Huang, “A fully parallel mixed-radix conversion algorithm for residue number applications,” *IEEE Transactions on Computers*, C-32(4), 398–402, 1983. doi:10.1109/TC.1983.1676242.
- [4] P. Ghayour, “A new algorithm to compare the magnitude of two RNS numbers,” arXiv:1612.09168, 2016. <https://arxiv.org/abs/1612.09168>.
- [5] M. Babenko, M. Deryabin, S. J. Piestrak, P. Patronik, N. Chervyakov, A. Tchernykh, and A. Avetisyan, “RNS number comparator based on a modified diagonal function,” *Electronics*, 9(11), 1784, 2020. doi:10.3390/electronics9111784.
- [6] M. Selianinau and Y. Povstenko, “An efficient parallel reverse conversion of residue code to mixed-radix representation based on the Chinese remainder theorem,” *Entropy*, 24(2), 242, 2022. doi:10.3390/e24020242.
- [7] L.-S. Didier, L. Glandus, N. El Mrabet, and J.-M. Robert, “Residue number system comparison revisited, a software perspective,” arXiv:2605.18415v1, 18 May 2026. <https://arxiv.org/abs/2605.18415>. This reference identifies the arXiv version; its internal manuscript date is 11 May 2025.
- [8] A. R. Luguterah, M. I. Daabo, S. Akobre, and M. A. Agebure, “Magnitude comparison and sign detection in residue number system,” *Asian Journal of Research in Computer Science*, 19(3), 144–161, 2026. doi:10.9734/ajrcos/2026/v19i3841.

A Reference algorithms and reproduction

The following pseudocode specifies the arithmetic independently of the test harness. The supplied Python source is authoritative for the reported timings.

```

SETUP(p1, p2, p3):
    require pairwise coprime moduli, each >= 2
    gamma = inverse(p3, p2)
    alpha = inverse(p2*p3, p1)
    beta = (alpha*p3) mod p1
    return gamma, alpha, beta

KEY(x1, x2, x3):
    i = (gamma*(x2-x3)) mod p2
    k = (alpha*(x1-x3)-beta*i) mod p1
    return (k, i, x3)

DIRECT(X_residues, Y_residues):
    return lexicographic_compare(KEY(X_residues), KEY(Y_residues))

DEFERRED(X_residues, Y_residues):
    kX = KEY(X_residues)[0]; kY = KEY(Y_residues)[0]
    if X_residues == Y_residues: return EQUAL
    if kX != kY: return compare(kX, kY)
    Z_residues[j] = (X_residues[j]-Y_residues[j]) mod p[j]
    return GREATER if KEY(Z_residues)[0] == 0 else LESS

```

The modulo operation must produce nonnegative remainders. Fixed-width translations must widen intermediate products before reduction; the Python code does not by itself certify overflow safety in C, Rust, Verilog, or VHDL.

For general least-to-most radices $m_1, \dots, m_n$, initialize $z_j = x_j$. At stage j , set $d_j = z_j$ and update every $\ell > j$ by $z_\ell \leftarrow \left[(z_\ell - d_j)m_j^{-1} \right]_{m_\ell}$. Inductively, the updated residues represent the quotient after removing the preceding digit. Thus

$$X = d_1 + m_1 d_2 + \dots + (m_1 \dots m_{n-1}) d_n,$$

with $0 \leq d_j < m_j$. This gives the standard MRC proof and operation count used above.

To reproduce the data and figures, run from the extracted project directory:

```

python experiments.py
python partition_experiment.py
python make_table.py
pdflatex -interaction=nonstopmode -halt-on-error manuscript.tex
pdflatex -interaction=nonstopmode -halt-on-error manuscript.tex

```

Python requires NumPy and Matplotlib for plotting; all residue computations use standard-library integers. Timing values will vary by machine and run. Assertions must remain enabled: do not run Python with `-O`. Data files record the seeds, environment, modulus triples, individual timing rounds, medians, and workload metrics. The PDF figures are generated from those computations and embedded without an external graphics service.

B What changed relative to the supplied drafts

This revision retains the corrected Euclidean proof, the canonical range contract, the MRC identification, and the direct comparison rule. It adds Theorem 4.1, the general prefix-collision identity, Theorem 5.1, Theorem 5.2, and Proposition 6.1, together with proofs and explicitly restricted objectives. The mathematical tools are elementary; presenting these derivations does not establish historical priority over all related literature.

The old finite-check counts were rerun and supplemented with larger inputs, exact shift-law checks, a partition-bound sweep, a skewed-distribution optimization, and an executed baseline timing study. Seven computed figures replace a purely textual experimental account. The longer draft’s unsimulated Verilog listing and abstract gate-count discussion are not presented as new experimental evidence. This revision also distinguishes expanded and factored arithmetic schedules and reports the CRT baseline’s favorable performance. It does not diagnose or claim to remedy an unknown preprint-screening decision.