

REIST-Division: Eine implementierungsorientierte Formulierung zentrierter Restarithmetik für die modulare Addition

Rudolf Stepan
Unabhängiger Forscher
ORCID: 0009-0004-2842-2579

Zusammenfassung—Zentrierte Restdarstellungen beschreiben vorzeichenbehaftete zyklische Korrekturen mit etablierten Restklassen. Dieser Bericht präzisiert das ganzzahlige Intervall und seine beschränkte modulare Additionsregel, einschließlich ungerader Moduli, Mittelpunktskonvention und Überlaufvoraussetzungen. Historische ARMv8-A- und x86-64-Messungen gegen den Restoperator werden dokumentiert. Sie wurden nach der Korrektur der Intervallgrenzen nicht erneut erhoben und belegen keinen Vorteil gegenüber beschränkter nicht-negativer modularer Addition, die ebenfalls ohne Division auskommt. Vor einer Leistungsbehauptung für die zentrierte Darstellung ist eine neue kontrollierte Evaluation nötig. Der Anwendungsbereich sind additionsdominierte Aktualisierungen; allgemeine Division und multiplikationsorientierte Montgomery- oder Barrett-Reduktion werden dadurch nicht ersetzt.

Index Terms—REIST, ganzzahlige Division, modulare Arithmetik, zentrierter Rest, balancierte Arithmetik, kryptographische Workloads, ARMv8-A, x86-64.

KORREKTURSTAND VOM 24. OKTOBER 2026

Die Tabellen stammen aus früheren Messungen. Sie wurden nach Korrektur der Intervallgrenzen nicht neu erhoben und belegen keinen Vorteil gegenüber optimierter nicht-negativer modularer Addition. Vor einer Journal-Einreichung sind korrigierte Kerne, genaue Quellstände und faire Vergleichsvarianten erneut zu messen.

I. EINLEITUNG

Die ganzzahlige Division legt eine Restkonvention fest. In C und C++ hat ein von null verschiedener Rest das Vorzeichen des Dividenten; für Aktualisierungen der Form $(a + b) \bmod m$ liegt damit häufig eine Division (oder ein vom Compiler erzeugter Divisionsersatz) auf dem kritischen Pfad. Balancierte Restdarstellungen sind mathematisch klassisch [2], werden jedoch meist als Konvention diskutiert und selten als konkretes Implementierungsmuster für schnelle modulare Addition behandelt.

Diese Arbeit stellt die REIST-Division als kompakte Formulierung vor: Modulare Werte werden in einem zentrierten Intervall dargestellt und über eine vorzeichenbehaftete Korrekturregel aktualisiert. Der Rest wird dabei nicht als passives Residuum, sondern als expliziter Korrekturterm verstanden, der Richtung und Betrag einer Abweichung kodiert. Diese Sichtweise liefert zugleich eine natürliche Modellierungssprache für Fehlerrückkopplung und Phasenausrichtung in Lo-

gistik, Scheduling, digitaler Signalverarbeitung, Robotik und hardwarenahen Korrekturschleifen.

A. Einordnung und Zielsetzung

REIST-Division lässt sich unmittelbar in hardwarenahem Ganzzahlcode umsetzen. Die zweite Hälfte dieser Arbeit behandelt REIST daher nicht nur als mathematisches Objekt, sondern als konkretes arithmetisches Primitiv. Berichtet werden Messungen aus einer quelloffenen Benchmark-Suite [4], die REIST und klassische Modulo-Operationen nebeneinander implementiert und auf ARMv8-A- sowie x86-64-Prozessoren auswertet.

Viele Beispiele stammen aus der gitterbasierten Kryptographie und der modularen Arithmetik; REIST ist jedoch keine kryptographiespezifische Konstruktion. Ihr Kernbereich ist die Ganzzahlarithmetik mit vorzeichenbehafteten Resten und die Modellierung zyklischer, rückgekoppelter Prozesse, in denen „zu viel“ und „zu wenig“ symmetrische Korrekturstände sind. Kryptographische Workloads dienen primär als Belastungstest und als ergiebige Quelle modulo-intensiver Operationen.

In hochoptimierten kryptographischen Bibliotheken wird die modulare *Multiplikation* üblicherweise durch Montgomery- oder Barrett-Reduktion beschleunigt. Die vorliegenden Benchmarks konkurrieren nicht mit diesen Verfahren. Der Vergleich erfolgt bewusst gegen den vom Compiler bereitgestellten Modulo-Operator, um den architektonischen Effekt des Ersatzes divisionsbasierter Restbehandlung durch eine vorzeichenbehaftete Korrekturregel zu isolieren – nicht, um Überlegenheit gegenüber modernsten modularen Reduktionsverfahren zu behaupten. Für multiplikationslastige Workloads ist REIST als komplementär zu Montgomery- und Barrett-Reduktion zu verstehen.

II. THEORETISCHER RAHMEN

A. Klassische Division

Für ganze Zahlen T und $B > 0$ liefert die klassische Division eindeutige ganze Zahlen q und r mit

$$T = qB + r, \quad 0 \leq r < B. \quad (1)$$

Diese Konvention bildet alle Abweichungen einseitig nicht-negativ ab und verankert damit eine Asymmetrie, die in zyklischen und rückgekoppelten Systemen suboptimal ist: Eine

kleine negative Korrektur (ein „Zurückspringen“) wäre häufig die effizienteste Handlung, wird durch das Intervall $[0, B)$ jedoch als große positive Korrektur dargestellt.

B. Definition der REIST-Division

Der REIST-Rest einer ganzen Zahl T bezüglich eines positiven Moduls B ist die eindeutige ganze Zahl r mit

$$T = qB + r, \quad q \in \mathbb{Z}, \quad -\frac{B}{2} \leq r < \frac{B}{2}. \quad (2)$$

REIST verwendet also ein zentriertes, halboffenes Restintervall $[-B/2, B/2)$. Eine praktische Konstruktion ist

$$r = ((T + \lfloor B/2 \rfloor) \bmod B) - \lfloor B/2 \rfloor, \quad (3)$$

wobei $\bmod$ den nicht-negativen Rest bezeichnet. Diese Konstruktion garantiert $r \in [-B/2, B/2)$; insbesondere tritt der Wert $+B/2$ bei geradem B nie auf, wodurch der Mittelpunktswert deterministisch aufgelöst ist.¹

Die Interpretation des Restes als Korrekturterm lautet:

- $r > 0$: das System liegt vor (Überschuss),
- $r < 0$: das System liegt zurück (Defizit),
- $r = 0$: exakte Ausrichtung.

C. Zusammenhang mit der Rundung

Aus der Intervallbedingung in (2) folgt unmittelbar für den Quotienten

$$q = \left\lfloor \frac{T}{B} + \frac{1}{2} \right\rfloor, \quad (4)$$

d.h. der REIST-Quotient ist die auf die nächste ganze Zahl gerundete rationale Zahl T/B , wobei Mittelpunktswerte (nur bei geradem B möglich) aufgerundet werden. Für ungerades B ist (4) exakt die eindeutige Rundung zur nächsten ganzen Zahl. REIST unterscheidet sich von reiner Rundungsarithmetik dadurch, dass der Rest als persistente vorzeichenbehaftete Korrekturvariable behandelt wird, die in iterative Aktualisierungsschleifen zurückgeführt werden kann.

D. Formale Eigenschaften

1) *Eindeutigkeit*: Für gegebene $T \in \mathbb{Z}$ und $B > 0$ existiert genau ein Paar (q, r) mit $T = qB + r$ und $r \in [-B/2, B/2)$: Das Intervall enthält genau B aufeinanderfolgende ganze Zahlen und damit aus jeder Restklasse modulo B genau einen Vertreter.

2) *Minimierung des Restbetrags*: Die Quotientenwahl (4) minimiert den Betrag des Restes: Für jede alternative Wahl $q' \in \mathbb{Z}$ gilt

$$|T - qB| \leq |T - q'B|. \quad (5)$$

Dies folgt direkt aus der Optimalität der Rundung zur nächsten ganzen Zahl. Die klassische Quotientenwahl $\lfloor T/B \rfloor$ minimiert den Restbetrag im Allgemeinen nicht.

¹Für ungerades B liefert (3) das symmetrische Intervall $[-(B-1)/2, (B-1)/2]$; der Mittelpunktswert $|r| = B/2$ kann dann nicht auftreten und die Darstellung ist stets eindeutig. Frühere Fassungen dieser Arbeit verwendeten das Intervall $(-B/2, B/2]$; die vorliegende Fassung legt einheitlich $[-B/2, B/2)$ fest, konsistent mit der Implementierung (3).

3) *Fehlerterm*: Mit $e = r/B$ gilt $|e| \leq 1/2$; die REIST-Division liefert damit die beste ganzzahlige Approximation des Quotienten T/B und übernimmt die Fehlerschranke der Rundungsarithmetik. Bei Gleichverteilung über die B ganzzahligen Vertreter gilt exakt $\text{Var}(r) = (B^2 - 1)/12$. Der Mittelwert ist für ungerades B null und für gerades B gleich $-1/2$; bei nicht-negativen Resten ist er $(B-1)/2$. Ein Mittelwert der Restdarstellung ist kein Nachweis dynamischer Drift. Deren Verlauf hängt von Eingangssignalen, Rückkopplung und Rundung ab.

4) *Additive Konsistenz*: Für $T_1 = q_1B + r_1$ und $T_2 = q_2B + r_2$ gilt

$$T_1 + T_2 = (q_1 + q_2)B + (r_1 + r_2), \quad r_1 + r_2 \in [-B, B). \quad (6)$$

Liegt $r_1 + r_2$ außerhalb des zentrierten Intervalls, genügt eine einzige Korrektur um $\pm B$, um in das Intervall zurückzukehren. Diese Eigenschaft ist die Grundlage der in Abschnitt IV beschriebenen schnellen modularen Addition.

5) *Beispiele*: Sei $T = 17$, $B = 10$: klassisch $17 = 1 \cdot 10 + 7$ ($r = 7$), REIST $17 = 2 \cdot 10 - 3$ ($r = -3$, minimaler Betrag). Sei $T = 28$, $B = 12$: klassisch wie REIST $r = 4$ (da $4 < 6$). Für $T = 34$, $B = 12$ wählt REIST $34 = 3 \cdot 12 - 2$ ($r = -2$), während klassisch $r = 10$ resultiert.

III. ABGRENZUNG ZU BESTEHENDEN VERFAHREN

Zentrierte Restsysteme (*balanced modulo, least absolute residue*) sind in der Zahlentheorie etabliert [2] und werden in NTT-Pipelines praktisch eingesetzt. Der Beitrag von REIST liegt nicht in einer neuen algebraischen Struktur, sondern in drei Punkten: (i) der konsequenten Behandlung des Restes als expliziter, persistenter Korrekturterm in Aktualisierungsschleifen, (ii) einer Implementierungsstrategie, die vollständig verzweigungsfrei (*branchless*) formulierbar und damit SIMD-freundlich ist, und (iii) einer systematischen empirischen Evaluation dieses Musters auf aktuellen CPU-Architekturen.

Tabelle I ordnet REIST in die Landschaft bestehender Verfahren ein. Barrett- und Montgomery-Reduktion adressieren primär die modulare *Multiplikation* und bleiben dafür die Verfahren der Wahl; REIST adressiert additionsdominierte Aktualisierungen. *Balanced modulo* beschreibt dieselbe Restklasse als nachträgliche Transformation des klassischen Restes, während REIST das zentrierte Intervall direkt über die Quotientenwahl bzw. die Korrekturregel erzeugt und den Rest als Zustandsvariable weiterführt.

Tabelle I
EINORDNUNG DER REIST-DIVISION GEGENÜBER BESTEHENDEN VERFAHREN.

Verfahren	Restintervall	Kernoperationen	Branchless
REIST	$[-B/2, B/2)$	ADD, CMP, MASK	ja
Restoperator	sprachabhängig	DIV bzw. Ersatz	teilweise
Beschränkt positiv	$[0, B)$	ADD, CMP, SUB	möglich
Barrett	$[0, B)$	MUL, SHR	ja
Montgomery	$[0, B)$	MUL, ADD	ja

A. Verhältnis zur Compiler-Optimierung

Compiler können Divisionen durch konstante Zahlen durch Multiplikation und Schieben ersetzen. Die hier relevante Vermeidung der Restoperation beruht jedoch auf beschränkten Operanden: Für $0 \leq a, b < B$ benötigt auch eine nicht-negative Darstellung nach $s = a + b$ nur die bedingte Subtraktion von B . Zentrierte und nicht-negative Varianten müssen mit denselben Bereichsannahmen verglichen werden. Beide funktionieren auch für einen zur Laufzeit bekannten positiven Modul. Die erzeugten Instruktionen sind am konkreten Binärcode zu prüfen.

IV. IMPLEMENTIERUNG UND ARCHITEKTURASPEKTE

A. Schnelle modulare Addition

Für ganze Zahlen $B > 0$ seien $l = -\lfloor B/2 \rfloor$ und $h = \lfloor (B-1)/2 \rfloor$. Sind $r_1, r_2 \in [l, h]$, genügt nach $s = r_1 + r_2$ genau eine der folgenden Korrekturen:

$$r = \begin{cases} s - B, & s > h, \\ s + B, & s < l, \\ s, & \text{sonst.} \end{cases} \quad (7)$$

Für gerade B kann die Summe $-B$ erreichen; das Summenintervall ist daher $[-B, B)$. Für beliebige Eingaben reicht eine Korrektur nicht. Alle Zwischenwerte müssen im verwendeten Ganzzahltyp darstellbar sein; die mathematische Verschiebungsformel ist ohne Überlaufprüfung kein allgemeiner C-Ausdruck. Masken können die Korrektur ausdrücken, garantieren aber ohne Prüfung des erzeugten Codes weder Verzweigungsfreiheit noch Konstantzeit.

B. Mikroarchitektonisches Kostenmodell

Beschränkte modulare Addition kann eine Restoperation durch Addition, Vergleich und Korrektur ersetzen. Das gilt sowohl für zentrierte als auch für nicht-negative Vertreter und lässt sich mit geeigneten SIMD-Instruktionen umsetzen. Latenz, Durchsatz, Registerdruck und Verzweigungswahl hängen von Zielprozessor, Compiler, Konstanten und Datenfluss ab. Eine allgemeine Ungleichung $C_{\text{REIST}} < C_{\text{MOD}}$ folgt daraus nicht. Erforderlich sind Messungen gegen beide beschränkten Varianten sowie gegen den Restoperator.

V. ANWENDUNGSDOMÄNEN

Über die hardwarenahe Arithmetik hinaus liefert die Interpretation des Restes als vorzeichenbehafteter Korrekturterm eine natürliche Modellierungssprache für mehrere Domänen:

- **Lieferketten und Logistik:** Ein positiver Rest entspricht einem Überbestand, ein negativer einem Unterbestand. Sicherheitsbestände lassen sich als symmetrische Puffer definieren; Losgrößen können in beide Richtungen angepasst werden.
- **Zyklische Systeme und Scheduling:** Ein negativer Rest kodiert Verzögerung, ein positiver Verfrühung. Rückwärts gerichtete Korrekturen in Produktions- oder Bewegungszyklen drücken eine vorzeichenbehaftete zyklische Abweichung aus; eine Driftreduktion folgt daraus allein nicht.

- **Digitale Signalverarbeitung:** Phasenfehler in Phasenregelschleifen (PLL) sind inhärent zyklisch und vorzeichenbehaftet [5]. Das zentrierte Intervall ist als Fenster für vorzeichenbehaftete Phasenfehler geeignet; Stabilität und Drift sind getrennt zu untersuchen.
- **Regelungstechnik und Robotik:** Rückgekoppelte Systeme (z. B. PID-Regler) arbeiten mit vorzeichenbehafteten Fehlern [3]. Für zyklische Größen (z. B. Winkelencoder) wählt REIST stets den kürzesten Korrekturweg anstelle einer erzwungenen positiven Rotation.
- **Prädiktive Modellierung:** Wird der Rest als eingebetteter Fehlerkorrekturterm mitgeführt, ist eine Verringerung von Bias gesondert am jeweiligen Rückkopplungsmodell nachzuweisen.
- **Ressourcenverteilung:** In verteilten Systemen erfasst ein negativer Rest Unterallokation explizit als Zustand und ermöglicht symmetrisches Load-Balancing.

VI. EXPERIMENTELLER AUFBAU

Berichtet werden Ergebnisse einer quelloffenen Benchmark-Suite [4], die eine klassische Modulo-Referenz und REIST-basierte Aktualisierungsregeln nebeneinander implementiert. Gemessen wurde auf zwei Architekturen:

- ARMv8-A (aarch64), mit SIMD-Varianten (NEON), so weit anwendbar;
- x86-64 (Intel Core i9-14900K), mit SIMD-Varianten, soweit anwendbar.

Alle Messungen erfolgten mit Compiler-Optimierung ($-O3$); SIMD-Spalten bezeichnen explizit vektorisierte Varianten. Die Benchmarks umfassen:

- 1) modulare Additionszähler ($x_{n+1} = (x_n + c) \bmod B$),
- 2) polynomielle modulare Addition (koeffizientenweise $z_i = (a_i + b_i) \bmod q$, der dominierende Kern additiver Schritte gitterbasierter Kryptographie, etwa in Kyber und Dilithium),
- 3) reine Restberechnung $x \bmod m$,
- 4) ARX-Workloads (ADD-ROTATE-XOR, z. B. ChaCha-artig),
- 5) Hash-Mixing-Workloads, die Multiplikation und Modulo kombinieren.

Angegeben werden Speedup-Bereiche (Laufzeit klassisch / Laufzeit REIST) über alle getesteten Moduli.

VII. ERGEBNISSE

Die Tabellen II und III fassen die Speedup-Bereiche zusammen.

Tabelle II
ARMV8-A: SPEEDUP-BEREICHE ÜBER DIE GETESTETEN MODULI.

Workload	O3	SIMD
Modulare Addition	2,1–2,3×	4,1–4,4×
Polynomielle modulare Addition	4,3–4,5×	4,3–4,5×
Nur Restberechnung	$\approx 1,0\times$	$\approx 1,0\times$
Hash-Mixing	0,79–0,80×	0,79–0,83×

Tabelle III
X86-64: SPEEDUP-BEREICHE ÜBER DIE GETESTETEN MODULI.

Workload	O3	SIMD
Modulare Addition	7,9–8,9×	7,1–8,7×
Polynomielle modulare Addition	12,4–12,8×	13,1–17,0×
Nur Restberechnung	$\approx 1,0\times$	$\approx 0,9\times$
Hash-Mixing	0,80–0,81×	0,73–0,75×

A. Interpretation

Über beide Architekturen zeigt sich ein konsistentes Muster:

- **Deutliche Gewinne**, wenn modulare Addition bzw. polynomielle modulare Addition dominiert. Dies ist konsistent mit dem Wegfall divisionsbasierter Restbehandlung und der lane-weisen SIMD-Korrektur.
- **Kein Vorteil** bei reiner Restberechnung – erwartungsgemäß, da REIST nicht dazu bestimmt ist, die Division zur Erstberechnung von $x \bmod m$ zu ersetzen.
- **Keine Veränderung** bei ARX-Workloads, die keine explizite Restdivision enthalten.
- **Verlangsamungen**: Ein Speedup von 0,73–0,83 entspricht etwa 20–37 % längerer Laufzeit ($1/S - 1$). Die Ursache ist durch die Tabellen allein nicht isoliert; eine Änderung statistischer Diffusion ist damit nicht nachgewiesen.

VIII. DISKUSSION UND LIMITATIONEN

REIST ist als praktische Formulierung zentrierter Restarithmetik mit einer auf modulare Addition optimierten Implementierungsstrategie zu verstehen. Daraus ergeben sich klare Grenzen:

- **Kein Ersatz für Montgomery/Barrett**: Für multiplikationslastige modulare Arithmetik bleiben etablierte Reduktionsverfahren angemessen. Direkte Vergleiche mit hand-optimierten Montgomery-/Barrett-Implementierungen liegen außerhalb des Umfangs dieser Arbeit; die Referenz ist bewusst der Compiler-Modulo-Operator.
- **Bereichsannahmen**: Der schnelle Korrekturschritt setzt beschränkte Operanden voraus. Ein zur Laufzeit bekannter Modul ist möglich; bei einem Modulwechsel müssen die Zustände erneut in dessen Bereich gebracht werden.
- **Memory-bound-Schleifen**: In speicherbandbreitenlimitierten Schleifen sind arithmetische Einsparungen zweitrangig; die Speicherlatenz bestimmt die Gesamtleistung.
- **ARX-basierte Verfahren**: ARX-Algorithmen (ChaCha20, BLAKE, SipHash u. a.) verwenden bereits wortbreites Wrapping modulo einer Zweierpotenz ohne explizite Restdivision.
- **Kompatibilität**: REIST erzeugt andere Reste als die klassische Division. Bestehende Systeme, die explizit $r \in [0, B)$ erwarten, müssen angepasst werden; ebenso können Anwendungen mit bewusst asymmetrischen Fehlertermen (z. B. Clipping) andere Ergebnismuster zeigen.
- **Seitenkanäle**: Die Korrekturregeln sind verzweigungsfrei implementierbar; Konstantzeiteigenschaften hängen jedoch von der konkreten Implementierung ab und werden hier nicht beansprucht.

A. Zukünftige Arbeiten

Naheliegende Erweiterungen umfassen: (i) die automatische Erkennung REIST-gerechter Muster in Compilern (LLVM, GCC) zur Erzeugung verzweigungsfreier, SIMD-fähiger Signed-Remainder-Arithmetik; (ii) FPGA-Implementierungen, da die Korrekturlogik mit geringem LUT-Aufwand und ohne Dividerblock realisierbar ist; (iii) die Integration in Residue-Number-Systeme (RNS) und CRT-basierte Verfahren, in denen betragskleine Reste die Fehlerfortpflanzung reduzieren; (iv) REIST-basierte NTT-Varianten mit zentrierten Koeffizientenbereichen; sowie (v) eine formale Analyse des Langzeit-Driftverhaltens in Regelschleifen.

IX. SCHLUSSFOLGERUNG

Die REIST-Division fasst den Rest der ganzzahligen Division als bidirektionalen Korrekturterm statt als einseitiges Residuum auf. Durch die Zulassung beschränkter negativer Reste und die Zentrierung des Restintervalls um null wird eine unnötige Asymmetrie der klassischen Division beseitigt und die Ganzzahlarithmetik an rückgekoppelte Rechenmodelle angeglichen.

Die historischen Tabellen dokumentieren Implementierungsvergleiche gegen einen Restoperator. Für Leistungsbehauptungen der korrigierten Fassung sind neue Messungen einschließlich einer beschränkten nicht-negativen Vergleichsimplementierung erforderlich. REIST ist daher am treffendsten als gezieltes arithmetisches Primitiv für modulo-additionslastige Rechenkerne zu verstehen – als SIMD-freundliche Darstellung, die größere modulare Arithmetik-Pipelines ergänzt, nicht ersetzt.

LITERATUR

- [1] D. E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 3. Aufl. Reading, MA, USA: Addison-Wesley, 1997.
- [2] I. Niven, H. S. Zuckerman und H. L. Montgomery, *An Introduction to the Theory of Numbers*, 5. Aufl. New York, NY, USA: Wiley, 1991.
- [3] K. J. Åström und R. M. Murray, *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton, NJ, USA: Princeton University Press, 2012.
- [4] R. Stepan, Benchmark-Quellen: <https://github.com/rudolfstepan/reist-crypto-bench>
- [5] A. V. Oppenheim und R. W. Schaffer, *Discrete-Time Signal Processing*, 2. Aufl. Upper Saddle River, NJ, USA: Prentice Hall, 1999.