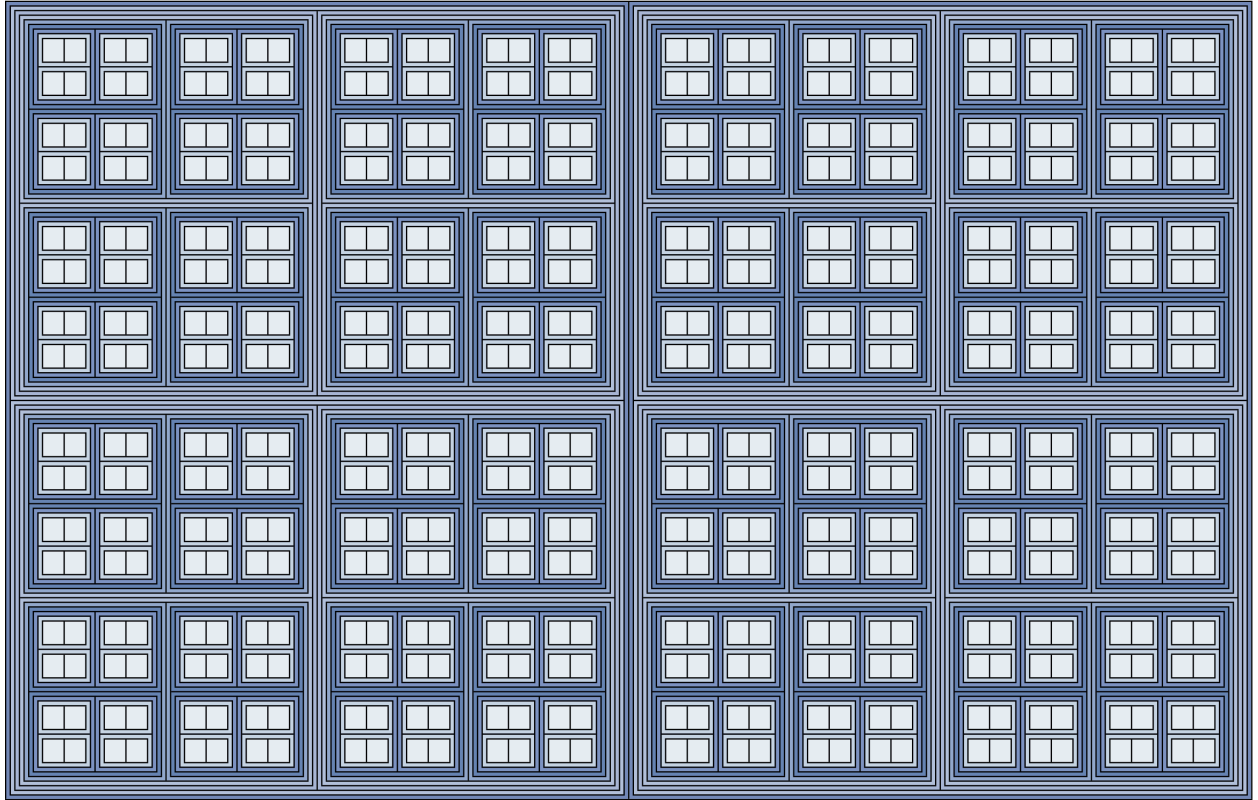


SciSci Research

サイサイ・リサーチ



# HENSEL CPU (ヘンゼル CPU): A 2-Adic Computing Architecture for Exact Arithmetic

James Douglas Boyd

**SciSci Research, Inc.**

Boulder, Colorado, United States

[www.sci-sci.org](http://www.sci-sci.org)

**Copyright** © 2025 by SciSci Research, Inc. All Rights Reserved.

**Citation Format:**

Boyd, J.D. (2025). HENSEL CPU (ヘンゼル): A 2-Adic Computing Architecture for Exact Arithmetic. SciSci Inventions, 1(1). DOI: 10.5281/zenodo.16374209

# Contents

<b>1</b>	<b>Introducing Hensel</b>	<b>2</b>
1.1	A CPU Architecture for $\mathbb{Q}_2$	2
1.1.1	Defining Exactness	2
1.1.2	The Question of Technical Risk	3
1.1.3	The Prize of Exact Arithmetic	3
1.2	Report Scope	4
<b>2</b>	<b>Novel Architectural and Coding Features</b>	<b>5</b>
2.1	Encoding Standards	5
2.1.1	FC 1-2025 Operand Encoding	5
2.1.2	FC 2-2025 Instruction Encoding	5
2.2	The Hensel Processor	5
2.2.1	2AALU Cluster	5
2.2.2	PR Cluster	5
<b>3</b>	<b>FC 1-2025</b>	<b>7</b>
3.1	A Quick Review of 2-adic Expansions and Coefficients	7
3.2	FC 1-2025 Encoding	7
3.3	Storing FC-1-2025-Encoded Numbers	9
<b>4</b>	<b>The Processor</b>	<b>11</b>
4.1	PR and 2AALU Clusters	11
4.2	Nested-Clustered Design	12
4.3	Cluster Load-Store and Addressing	12
<b>5</b>	<b>Parallelized Arithmetic</b>	<b>15</b>
5.1	Parallel Arithmetic in $\mathbb{Q}_2$ via Coefficient-Modification	15
5.2	FC 2-2025 Encoding	15
5.3	Parallelization and Non-Archimedean Distance	16
5.4	$\text{Max}(\Delta_2^{\tau_{\text{h}_\ell}})$ Parallelization	18
5.5	Constraints on Optimization	21
<b>6</b>	<b>Further Prospects</b>	<b>22</b>
6.1	Error Correction and Fault Tolerance	22
6.1.1	Error-Checking with $(\chi, \tau)$ -Payloads	22
6.1.2	$\Delta_2^\mu$ Trajectory-Defect Checks	22
6.1.3	$\Xi_{2\text{AALU}}$ Consensus	22
6.2	Supercomputing	23
<b>7</b>	<b>Appendix: Proofs of Hensel Performance Properties</b>	<b>25</b>

# 1 Introducing Hensel

## 1.1 A CPU Architecture for $\mathbb{Q}_2$

Introduced preliminarily in this report is the Hensel CPU (ヘンゼル CPU), designed according to a novel computing architecture with the aspiration of replacing floating-point arithmetic with exact arithmetic performed in  $\mathbb{Q}_2$  (i.e., 2-adic arithmetic). Here, exact arithmetic denotes arithmetic which, although subject to computational bounds (e.g., constrained by a given bit-width), is nonetheless performed on operands whose representation in bits is unique. The Hensel architecture is designed for arithmetic in  $\mathbb{Q}_2$ , rather than  $\mathbb{R}$ . In floating-point arithmetic, "floating-point numbers" (i.e., elements of  $\mathbb{Q}$ ) approximate elements of  $\mathbb{R}$  with finite precision. It has been known among some computer scientists since the 1970's that  $p$ -adic numbers (e.g., elements of  $\mathbb{Q}_2$ ) – where, from Ostrowski's theorem, we know that  $\mathbb{Q}_p$  and  $\mathbb{R}$  are the only completions of  $\mathbb{Q}$  – admit exact, unique representations with finite encodings. (Examples of precedents include the so-called "Hensel codes" of Krishnamurthy *et al.*, the work of Horspool-Hehner, and Doris' system for exact  $p$ -adic arithmetic in Magma.)

Unlike the above precedents, Hensel is an architecture, rather than an algorithm or software package. Thus, it is designed to perform arithmetic in  $\mathbb{Q}_2$  at the machine level, where  $\mathbb{Q}_2$  is distinct from all other  $\mathbb{Q}_{p>2}$  in that the coefficients of 2-adic expansions are  $a_i \in \{0, 1\}$  and thus can be written in bits. Descending to the architectural level carries several prospective advantages to CPU users. First, users can utilize a CPU performing arithmetic in  $\mathbb{Q}_2$  at the machine level without any familiarity with 2-adic arithmetic; operands in  $\mathbb{Q}_2$  can be expressed – for instance, symbolically – in user-recognizable form in higher-level programming languages. Thus, unlike software such as Magma, which offers exact arithmetic to number theorists familiar with  $\mathbb{Q}_p$ , the Hensel CPU is designed to bring the accu-

racy and performance of exactness to general users. Moreover, by developing a design to realize exact arithmetic at an architectural level, SciSci Research and its Future Computing group endeavor to confront the barriers posed by floating-point to accelerated computing, and help to realize an exact accelerated computing capability unhindered by tradeoffs between performance, accuracy, and cost.

### 1.1.1 Defining Exactness

Although no computing system can overcome the limitation of computing with finite resources over fields with cardinalities of the continuum, computers can be designed such that the operands over which they compute are unique; such is the aspiration of exactness pursued by SciSci Research and Future Computing in designing the Hensel CPU. Thus, the criterion of exactness is not to be mistaken for the promise of computability. One will ineluctably encounter examples of arithmetic over particular numbers that the Hensel CPU cannot perform within its bit-width (in which case one will receive an error message, rather than an approximation), but the architecture is designed such that, when it can perform arithmetic, it does so exactly. Furthermore, the architecture, which is designed to favor scaling towards supercomputing applications, is advanced with the aspiration of extending exactness to the largest collection of operands possible (e.g., with a large bit-width).

It should be noted that the Hensel CPU is not entirely untethered from the question of approximation insofar as its instructions and operand encodings are developed with the assistance of software such as Sage and Magma, which return so-called "lazy" representations of 2-adic numbers (i.e., up to a specified precision). Nonetheless, arithmetic performed by a Hensel CPU can nonetheless remain exact so long as its accepted operands and instructions are restricted to those with unique representations that can be encoded within the CPU bit-width.



### 1.1.2 The Question of Technical Risk

It should be emphasized that the novelty of the Hensel architecture resides not in an argument regarding the prospect of exact arithmetic, a critique of floating-point, or an observation that  $\mathbb{Q}_p$  can be used advantageously for exact arithmetic; such arguments can already be found in the literature. The novel value proposition of the Hensel CPU is found in its realization of 2-adic arithmetic at the hardware level. Although a rough design for the architecture is presented here, the technical risks of realizing novel hardware (e.g., arithmetic logical units and processor registers) remain outstanding. Nonetheless, it should be emphasized that an architecture for operands in  $\mathbb{Q}_2$ , whose expansion coefficients can be encoded in bits, should be compatible with extant MOSFET technology. Such compatibility significantly de-risks the Hensel CPU prospect relative to other computing paradigms such as quantum computing, in which case one must develop wholly new electronics, such as transistors, for computing with qubits. Thus, although the Hensel CPU will involve new hardware, it doesn't require a paradigmatic alternative to current electronics and semiconductor technology; it merely requires a new CPU that performs arithmetic in  $\mathbb{Q}_2$  with such technology.

### 1.1.3 The Prize of Exact Arithmetic

In floating-point arithmetic, real numbers (i.e., elements of  $\mathbb{R}$ ) are approximated by "floating-point numbers", which are rationals (i.e., elements of  $\mathbb{Q}$ ). Reals are given decimal representations, which are Cauchy sequences of rationals; in the case of floating-point, these are truncated to be of finite precision. Of course, given finite resources, representations must be finite. The coding-theoretic issue, in the case of  $\mathbb{R}$ , pertains to an analytic issue: real numbers don't have unique Cauchy sequences; they can only be given up to equivalence. Moreover,  $\mathbb{R}$  is in

fact a field of equivalence classes of Cauchy sequences. As a consequence, the accuracy limitations from which floating-point arithmetic suffers can be seen as a consequence of giving finite-precision representations of non-unique approximations of elements of  $\mathbb{R}$ . For instance, suppose, given a bit-width allowing 8 decimal places, one tries to approximate  $\frac{1}{3}$ . One cannot distinguish the approximation from  $\frac{33333333}{100000000}$ ; the representation of  $\frac{1}{3}$  is non-unique.

In the case of  $\mathbb{Q}_2$ , every 2-adic number has a unique 2-adic expansion, which is an infinite series  $\sum_{i=z}^{\infty} a_i 2^i$  (where  $z \in \mathbb{Z}$  and  $a_i \in \{0, 1\}$ ). Writing the coefficients of these series, we obtain unique binary representations. One can compute the exact values of 2-adic expansions of  $n \in \mathbb{Q}_2$  using the convergent properties of infinite series with respect to the 2-adic norm,

$$|n|_2 = 2^{-v_2(n)} \quad (1)$$

where  $v_2$  is the 2-adic valuation  $v_2 : \mathbb{Q} \rightarrow \mathbb{Z} \cup \infty$  (i.e.,  $\infty$  in the case of  $n = 0$ ). For instance, the following series for  $\frac{1}{3} \in \mathbb{Q}_2$  converges with respect to  $|\cdot|_2$ :

$$\begin{aligned} \frac{1}{3} &= \\ &1 + 2(1 + 2^2 + 2^4 + \dots) \\ &= 1 + \frac{2}{1 - 2^2} \end{aligned} \quad (2)$$

as is the case for  $\frac{1}{5} \in \mathbb{Q}_2$ :

$$\begin{aligned} \frac{1}{5} &= \\ &1 + 2^2(1 + 2^4 + 2^8 + \dots) + \\ &2^3(1 + 2^4 + 2^8 + \dots) \\ &= 1 + \frac{2^2}{1 - 2^4} \end{aligned} \quad (3)$$

As discussed in this report, in the case of the Hensel CPU, the crux of its value proposition is computation with finite, efficient encodings of 2-adic expansions in a manner that preserves uniqueness and hence, exactness.

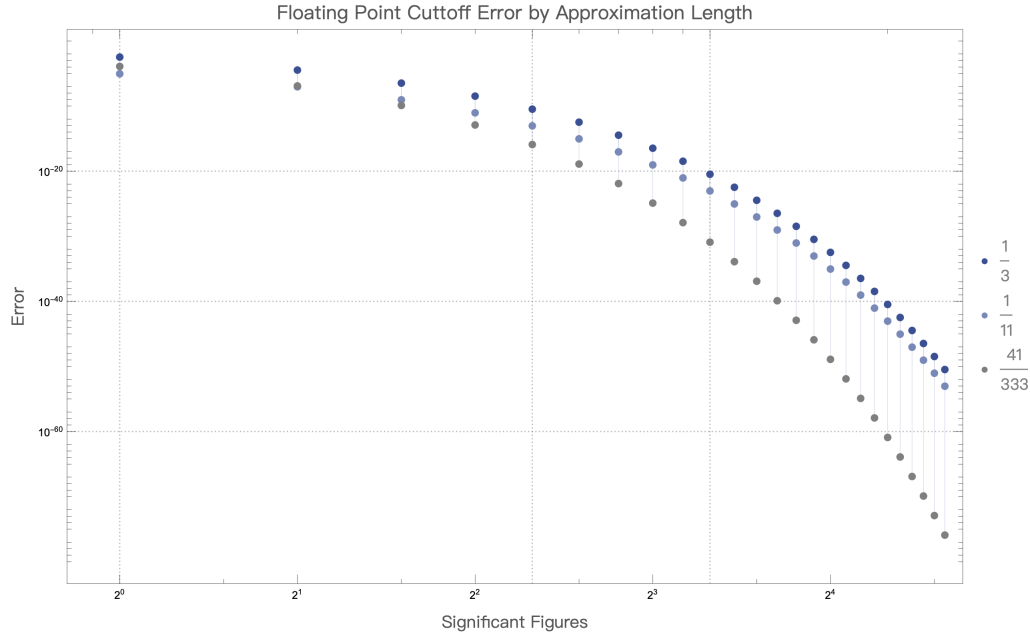


Figure 1: A plot of approximation errors for  $\frac{1}{3}$  (i.e.,  $\frac{1}{3} - \frac{3.3 \times 10^i}{10^{i+1}}$ ,  $i \in \{1, 3, \dots, 49\}$ ),  $\frac{1}{11}$  (i.e.,  $\frac{1}{11} - \frac{\sum_{k=0}^i 9 \times 10^{k+1}}{10^{i+3}}$ ,  $i \in \{2, 4, \dots, 50\}$ ), and  $\frac{41}{333}$  (etc.) as one increases  $i$ .

## 1.2 Report Scope

The most rudimentary objective of this report is to introduce how 2-adic numbers are to be stored and operated upon in the Hensel architecture. This requires a coding-theoretic description of 2-adic operands, a data-structural description of how they are stored, and a logical-functional description of how they are subject to arithmetic operations. In light of the novelty of a CPU architecture designed for  $\mathbb{Q}_2$ , it is necessary to proffer this description in a manner that begins with  $p$ -adic analysis and proceeds to computer engineering. With the intersection of these disciplines relatively empty, it is necessary to begin at a fundamental level, describing operands and operations at the level of lists and functions. Doing so will involve relatively simple mathematics but a moderately sophisticated regime of notation for assigning mathematical descriptions to the architectural components of the CPU.

The Hensel CPU architecture is designed for

a load-store instruction-set architecture (ISA). This first report is intended to preliminarily introduce the architectural features, especially Hensel processor components, whose load-store roles are most integral. Thus, the report will dedicate particular attention to process registers and arithmetic logical units, as well as the architectural properties with which their design is endowed for 2-adic computing. This report intends to describe their design and function, as well as characterize (and provide mathematical proofs of) properties anticipated to be of utility in accelerated computing.

Looking ahead, beyond the rudimentary presentation given here, more rarefied architectural descriptions – including ISA, microarchitecture, and implementation – will each be given their own subsequent reports. This report is the first of what will be a Hensel series by SciSci Research and its Future Computing group.

## 2 Novel Architectural and Coding Features

This report introduces, in addition to several components of the Hensel CPU architecture, standards used by SciSci and Future Computing for representing 2-adic numbers and instructions (comparable in purpose to standards given for floating-point arithmetic, such as IEEE 754-1985). These standards are necessarily internal, rather than industrial; that is to say, they are used internally by SciSci and Future Computing as a coding-theoretic basis for architectural design specifications.

### 2.1 Encoding Standards

#### 2.1.1 FC 1-2025 Operand Encoding

This report introduces FC 1-2025 (Future Computing Standard 1-2025 for 2-adic Arithmetic), a coding standard for 2-adic numbers. FC 1-2025 gives an efficient coding of the coefficients of 2-adic expansions, and thus provides an exact, non-approximate encoding of 2-adic numbers (*pace* the  $\mathbb{R}$ -approximations found indelibly in floating-point arithmetic). That is to say, all  $n \in \mathbb{Q}_2$  whose FC 1-2025 encoding is within the bit-width of the CPU can be subject to exact arithmetic. (As discussed in Section 6.2, this bit-width can be very large.)

#### 2.1.2 FC 2-2025 Instruction Encoding

The second standard introduced in this report is FC 2-2025, a standard for encoding parallelizable arithmetic instructions for executing operations on FC-1-2025-encoded 2-adic numbers.

### 2.2 The Hensel Processor

Hensel's arithmetic logical units (ALUs) and processor registers (PRs) are designed according to a novel, nested framework, according to which a moderate number of low-cost ALUs and PRs are assembled into a cluster, with two key prospective advantages (one afforded by nesting and the other by clustering). First, the nested structure is utilized for optimizing storage of FC-1-2025-encoded operands in the PRs and execution of FC-2-2025-encoded instructions by ALUs due to the correspondence between this nested structure and the relationships between 2-adic numbers. (For instance, note that the distance between 2-adic numbers is non-archimedean; rather than forming a "number line" like the case of  $\mathbb{R}$ , it forms a nested structure.) Second, clustering allows for parallelization of the optimal storage and computing capabilities made available by nested design.

#### 2.2.1 2AALU Cluster

2-adic arithmetic logical units (2AALUs) are arithmetic logical units that perform arithmetic in  $\mathbb{Q}_2$ . Together, the 2AALUs belong to the 2AALU cluster  $\Xi_{2AALU}$ , which can perform arithmetic in parallelized fashion.

#### 2.2.2 PR Cluster

Processor registers are also designed in a cluster,  $\Xi_{PR}$ . Operands are stored in the processor (and main memory) in FC 1-2025 format according to a triple-tree convention discussed in this report. The individual PRs in  $\Xi_{PR}$ , coordinating with the load-store unit (LSU), are instructed by the control unit (CU) to supply operands to  $\Xi_{PR}$  for parallelized execution and to receive outputs.

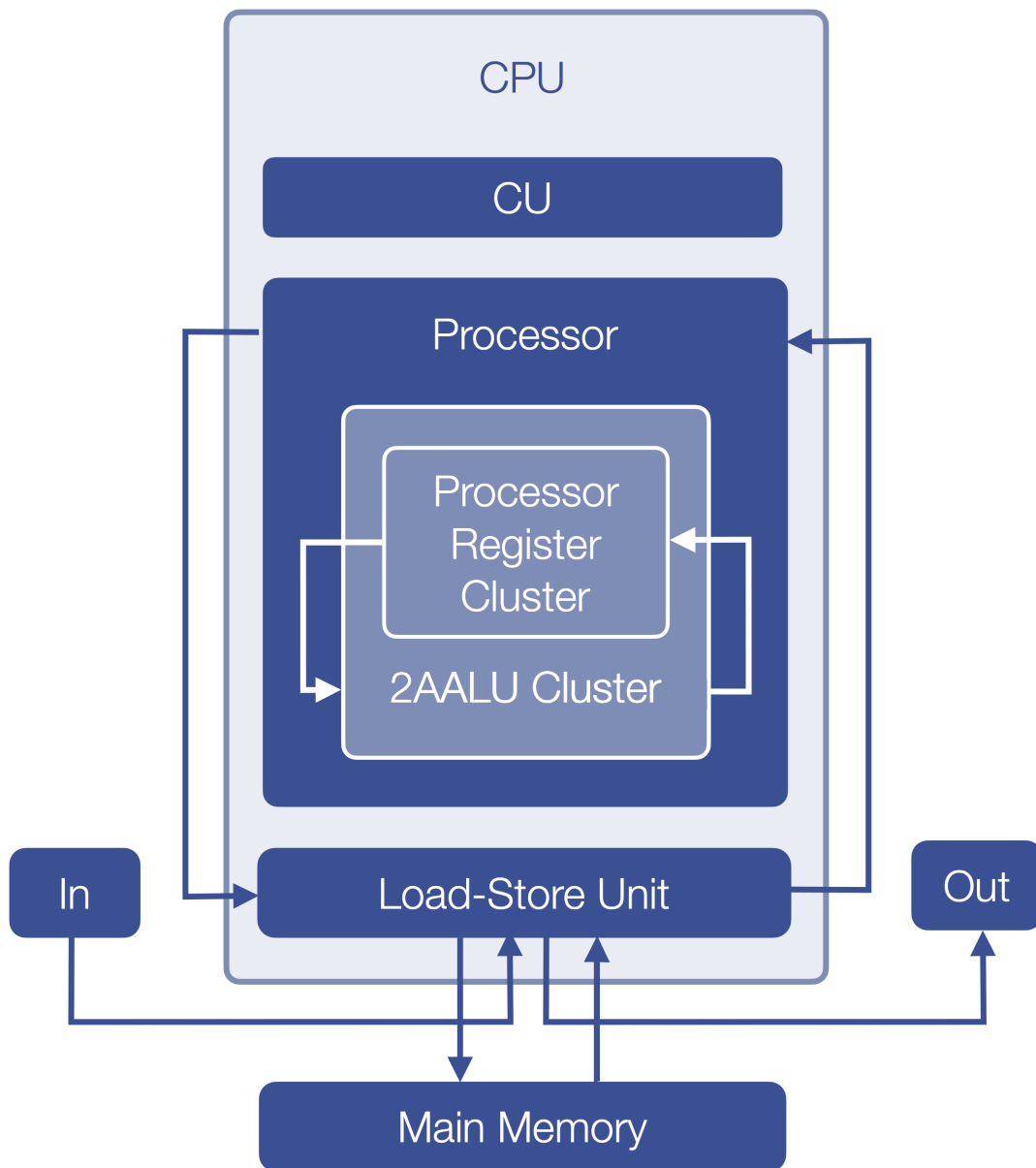


Figure 2: Block diagram of the Hensel CPU, processor, control unit (CU), load-store unit (LSU), and main memory, highlighting 2AALU and processor register cluster relations. (For simplicity of presentation, in- and outgoing arrows with respect to the the CU are omitted.)

### 3 FC 1-2025

#### 3.1 A Quick Review of 2-adic Expansions and Coefficients

Recall from  $p$ -adic analysis that a 2-adic number ( $n \in \mathbb{Q}_2$ ) admits the following expansion

$$n := \sum_{i=z}^{\infty} a_i 2^i, \quad a_i \in \{0, 1\}, \quad z \in \mathbb{Z} \quad (4)$$

These unique expansions give exact sums owing to the convergent properties of their series with respect the 2-adic norm. For instance, in the case of the following,

$$a(2^i + 2^{i+k} + 2^{i+2k} + \dots) = \frac{a}{1 - 2^k} \quad (5)$$

whereas, in the case of  $\mathbb{R}$ , geometric series  $\sum_{k=0}^{\infty} a_k r^k$  only converge when  $|r| < 1$ , it is the case that 2-adic expansions converge with respect to  $|r|_2$  even when  $|r| \geq 1$ . Properties such as this, as well as the uniqueness of 2-adic expansions, provide a basis for exact arithmetic.

For purposes of introducing the FC 1-2025, we turn our attention to the coefficients  $a_i$ . Let's revisit the example of  $\frac{1}{3} \in \mathbb{Q}_2$ . The first 20 summands (including 0 summands) in its 2-adic expansion are

$$\frac{1}{3} = 1 + 2 + 2^3 + 2^5 + 2^7 + 2^9 + 2^{11} + 2^{13} + 2^{15} + 2^{17} + 2^{19} + \dots \quad (6)$$

(Here, the coefficient for  $2^3$ , for instance, is 1, whereas the coefficient for  $2^4$ , for instance, is 0.) We then write the coefficients from right to left, with an overbar over repeating coefficients. In this case,

$$\frac{1}{3} = \overline{011}1.2 \quad (7)$$

The first 20 summands in the expansion of  $\frac{1}{5}$  are

$$\frac{1}{5} = 1 + 2^2 + 2^3 + 2^6 + 2^7 + 2^{10} + 2^{11} + 2^{14} + 2^{15} + 2^{18} + 2^{19} + \dots \quad (8)$$

which can be abbreviated as

$$\frac{1}{5} = \overline{0011}01.2 \quad (9)$$

Abbreviations of this kind are encoded according to the FC 1-2025 standard as follows.

#### 3.2 FC 1-2025 Encoding

We begin by taking the coefficients  $a_i$  of a 2-adic expansion sequence of a given  $n \in \mathbb{Q}_2$ , which will be ordered from right to left. The coefficient-list-form  $S$  of the sequence for the 2-adic expansion of  $n \in \mathbb{Q}_2$  is as follows:

$$S(n) = (a_{\infty}, \dots, a_1, a_0, \dots, a_{z+1}, a_z) \quad (10)$$

The FC 1-2025 encoding of a given  $n \in \mathbb{Q}_2$  consists of finite sublists of  $S(n)$  which still give a unique encoding:

$$FC(S(n)) := (\perp, R_{FC}, \perp, L_{FC}, \perp, T_{FC}) \quad (11)$$

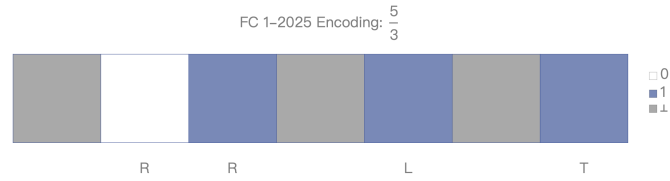
$R_{FC}$  is the sublist of repeating coefficients in the 2-adic expansion (e.g.,  $\overline{0011}$  for  $\frac{1}{5}$ ). (Infinitely repeated 0s – e.g., for integers – are omitted.)  $L_{FC}$  is the sublist of non-repeating coefficients to the left of the "decimal" point (e.g., 01 for the case of  $\frac{1}{5}$ ), and  $T_{FC}$  is the sublist of coefficients for summands with negative exponents, typically written to the right of the "decimal point" (e.g., 1 in the case of  $\frac{1}{2}$ , which is written as .1<sub>2</sub>). (T is minimized in FC-1-2025, such that superfluous 0s are not added. In many cases, such as  $\frac{1}{3} \in \mathbb{Q}_2$ , it is the empty; for instance,  $T_{FC(S(\frac{1}{3}))} = ()$ .)  $\perp$  is the "no operation" symbol separating R, L, and T. For instance,

$$FC\left(S\left(\frac{1}{3}\right)\right) := (\perp, (0, 1), \perp, (1), \perp, ()) \quad (12)$$

To take another example,

$$FC\left(S\left(\frac{1}{5}\right)\right) := (\perp, (0, 0, 1, 1), \perp, (0, 1), \perp, ()) \quad (13)$$

Another important example is as follows: by convention,  $FC(S(0)) := (\perp, \perp, (0), \perp)$ . Visual illustrations of several FC 1-2025 encodings are given in Figure 3 and Figure 4.

Figure 3: Array plot visualization (with color legend) of the FC 1-2025 Encoding of  $\frac{5}{3}$ .

Rational	2-adic Expansion	2-adic Number	FC2 1-2025 Encoding
$\frac{1}{2}$	$2^{-1} = \frac{1}{2}$	$0.1_2$	
$\frac{1}{5}$	$1 + 2^2(1 + 2^4 + 2^8 + \dots) + 2^3(1 + 2^4 + 2^8 + \dots) = 1 + \frac{2^2}{1-2^4} + \frac{2^3}{1-2^4} = \frac{1}{5}$	$\overline{00}\overline{11}01_2$	
$\frac{7}{10}$	$2^{-1} + 1 + 2^2(1 + 2^4 + 2^8 + \dots) + 2^3(1 + 2^4 + 2^8 + \dots) = 2^{-1} + 1 + \frac{2^2}{1-2^4} + \frac{2^3}{1-2^4} = \frac{7}{10}$	$\overline{00}\overline{11}01.1_2$	
$\frac{1}{3}$	$1 + 2(1 + 2^2 + 2^4 + \dots) = 1 + \frac{2}{1-2^2} = \frac{1}{3}$	$\overline{0}\overline{1}1_2$	
$\frac{3}{7}$	$1 + 2^2(1 + 2^3 + 2^6 + 2^9 + \dots) = 1 + \frac{2^2}{1-2^3} = \frac{3}{7}$	$\overline{00}\overline{1}01_2$	
$\frac{22}{7}$	$2 + 2^3 + 2^4(1 + 2^3 + 2^6 + \dots) + 2^5(1 + 2^3 + 2^6 + \dots) = 2 + 2^3 + \frac{2^4}{1-2^3} + \frac{2^5}{1-2^3} = \frac{22}{7}$	$\overline{0}\overline{1}\overline{1}1010_2$	

Figure 4: Table of six 2-adic numbers, their 2-adic expansions, and FC 1-2025 encodings.

### 3.3 Storing FC-1-2025-Encoded Numbers

FC-1-2025-encoded 2-adic numbers are stored in a "triple-tree" format. We'll define a triple tree as a connected graph consisting of three binary trees  $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$  whose parent nodes  $v_0^{B_k}$  are glued to a vertex  $v_\perp$ . Thus, appended to the edge list  $E_k$  of each is an undirected edge between the parent vertex  $v_0^{B_k}$  and  $v_\perp$ . Given  $\mathcal{B}^* := (E^*, V)$ , where  $E_1^* = E_1 \cup (v_0^{B_1}, v_\perp)$ ,  $E_2^* = E_2 \cup (v_0^{B_2}, v_\perp)$ , and  $E_3^* = E_3 \cup (v_0^{B_3}, v_\perp)$ , we define our triple-tree  $\mathcal{T}$  as a clique-sum (shown in Figure 5):

$$\mathcal{T} := \bigcup_{k=1}^3 \mathcal{B}_k^* \quad (14)$$

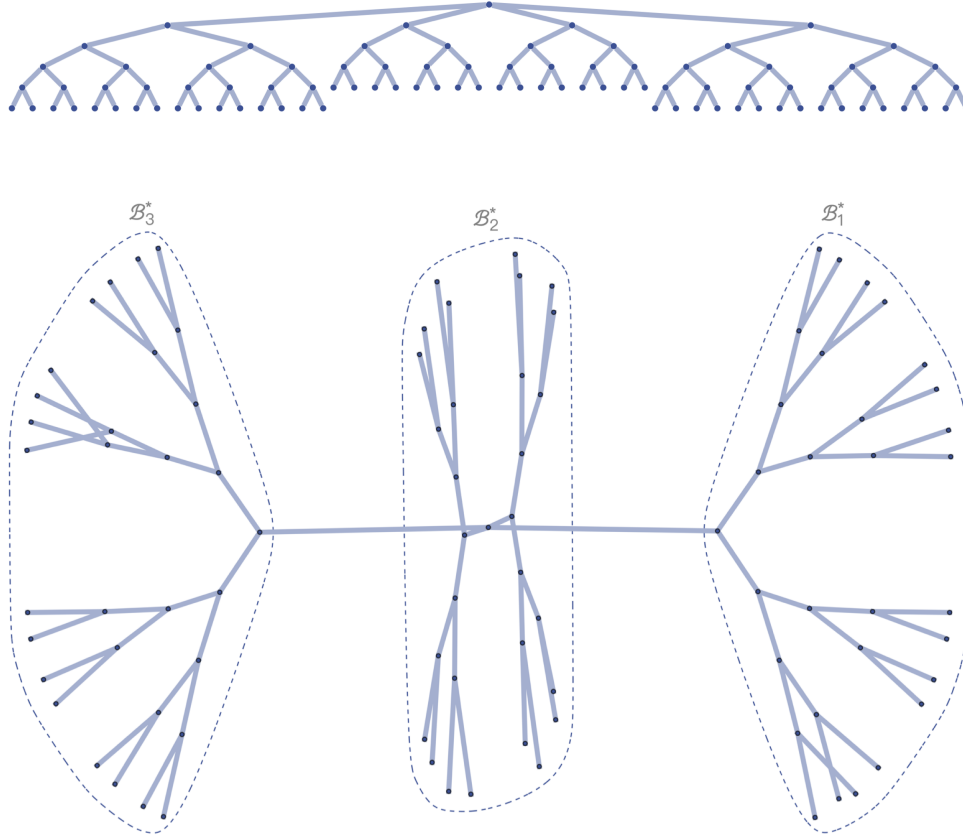


Figure 5: Top: an example illustration of  $\mathcal{T}$ . Bottom: the same illustration of  $\mathcal{T}$  highlighting  $\mathcal{B}_1^*$ ,  $\mathcal{B}_2^*$ , and  $\mathcal{B}_3^*$ . Note that these graph embeddings, for simplicity, collapse  $v_\perp$  onto  $v_0^{B_2}$ .

With respect to the FC 1-2025 format,  $R_{FC(-)}$  data are stored as  $\mathcal{B}_1^*$ ,  $L_{FC(-)}$  as  $\mathcal{B}_2^*$ , and  $T_{FC(-)}$  as  $\mathcal{B}_3^*$ . ( $\mathcal{T}$ -form examples for FC ( $S(\frac{7}{12})$ ), FC ( $S(\frac{9}{20})$ ), and FC ( $S(\frac{47}{60})$ ) are shown in Figure 6.) Binary tree encoding assigns each  $a_i$  in  $R_{FC}$ ,  $L_{FC}$ , or  $T_{FC}$  to a vertex  $v_i \in \mathcal{B}_k^*$ , beginning with  $v_0^{B_k}$ , with the coefficient assigned thereto indexed as  $a_0$ . Thus, one can think of  $R_{FC}$ ,  $L_{FC}$ , or  $T_{FC}$  as each stored in a  $\mathcal{B}_k^*$  as its own 2-adic number (beginning to the left of the "decimal point" at  $a_0$ ), though nevertheless together giving an FC-1-2025 encoding as a triple-tree data-structure. Beginning each  $\mathcal{B}_k^*$  encoding with  $a_0$  eliminates the need to encode  $T_{FC}$  using negative indices, which is of consequence for  $\max(\Delta_2^\mu)$  parallelization, as discussed subsequently.

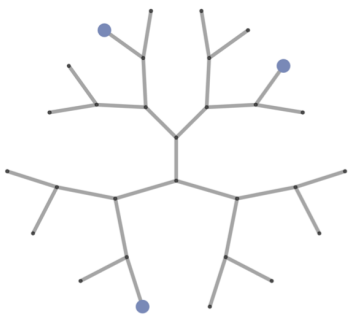

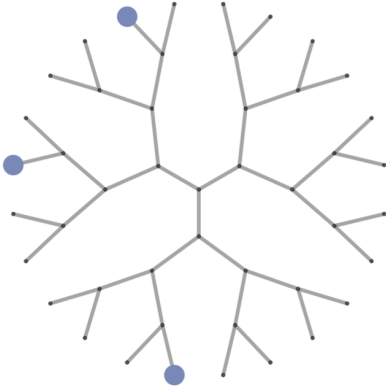

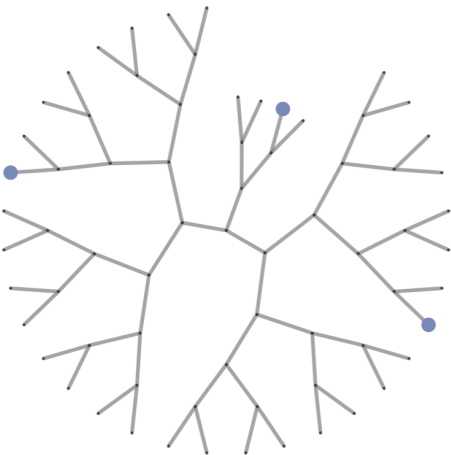

Rational	Triple-Tree Form	FC 1-2025 Encoding
$\frac{7}{12}$	 A fractal-like tree structure with three blue dots at the ends of its branches.	 A horizontal bar with 12 segments, alternating between blue and white.
$\frac{9}{20}$	 A fractal-like tree structure with three blue dots at the ends of its branches.	 A horizontal bar with 20 segments, alternating between blue and white.
$\frac{47}{60}$	 A fractal-like tree structure with three blue dots at the ends of its branches.	 A horizontal bar with 60 segments, alternating between blue and white.

Figure 6: Triple-tree plots for three FC-1-2025-encoded 2-adic numbers. (Here, the embeddings separate  $v_{\perp}$  from the  $v_0^{B_i}$ .)



## 4 The Processor

### 4.1 PR and 2AALU Clusters

The Hensel CPU architecture adopts a novel approach to the design of its processor, which is a cluster composed of smaller units (i.e., small, low-cost 2AALUs and PRs), belonging to one of two main clustered components: a processor register cluster  $\Xi_{\text{PR}}$  and a 2AALU cluster  $\Xi_{2\text{AALU}}$ . Operands are stored in and retrieved from  $\Xi_{\text{PR}}$ , and arithmetic logic is performed in  $\Xi_{2\text{AALU}}$ . The processor cluster has a nested design, with the  $\Xi_{\text{PR}}$  at the innermost level enclosed in nested layers of 2AALUs (shown in Figure 7 and Figure 8). Here, the inner-to-outer positions of nest levels correspond to the right-to-left positioning of  $a_i$  in FC 1-2025 encodings of  $S(n)$ , such that 2AALUs positioned at different levels can modify different  $a_i$ , and, doing so simultaneously at different levels, parallelize  $\Xi_{2\text{AALU}}$  operations. This is achieved via execution of FC-2-2025-encoded instructions.

Hensel's nested-clustered processor design is chosen to meet the objective of optimally storing and efficiently computing with numbers in  $\mathbb{Q}_2$  via an architecture that is commensurate with 2-adic arithmetic. For instance, the Hensel processor efficiently maximizes 2-adic output-operand distance over minimal operations, precisely because its nested structure is commensurate with the non-archimedean nature of distance between 2-adic numbers, which, contrary to the archimedean case of  $\mathbb{R}$ , is not measured with respect to a line, but instead a non-archimedean distance that gives a nested structure; the PR and 2AALU clusters are designed according to this structure.

Given the  $\mathcal{T}$ -form of a given  $\text{FC}(S(n))$ , the  $\mathcal{B}_k^*$  are individually stored in  $\Xi_{\text{PR}}$ , where, to be more precise,  $\Xi_{\text{PR}}$  consists of a master PR  $\mathcal{M}_{\text{PR}}$  adjacent to clustered PRs  $\mathcal{C}_{\text{PR}}$ . Together, they comprise the PR cluster, which we write in notation as a set of PRs  $\Xi_{\text{PR}} := \left( \bigcup_{i=1}^{\mathcal{N}} \mathcal{C}_{\text{PR}_i} \right) \cup \mathcal{M}_{\text{PR}}$  (where  $\mathcal{N} < 2^B$ , with  $B$  being the architecture bit-width). Each  $\mathcal{C}_{\text{PR}}$

has an address for storing a particular  $\mathcal{B}_k^*$  according to a  $\chi$ -address system described subsequently, and is loaded via activation by an operational bit  $\pi$  from  $\mathcal{M}_{\text{PR}}$  as prompted by the LSU. FC-1-2025 numbers are all stored, in distributed fashion, throughout the PR cluster, where particular operands are loaded by the LSU by activating the  $\mathcal{C}_{\text{PR}}$  bearing the address for a given  $R_{\text{FC}}$ ,  $L_{\text{FC}}$ , or  $T_{\text{FC}}$ . That is to say, for whatever fixed bit-width  $B$ , the load-store architecture for the Hensel CPU processor will be designed so that all  $R_{\text{FC}}$ ,  $L_{\text{FC}}$  and  $T_{\text{FC}}$  permissible within the bit-width are addressed to distinct, particular  $\mathcal{C}_{\text{PR}}$ . For instance, consider the operands  $\text{FC}(S(q))$  and  $\text{FC}(S(p))$ , and output  $\text{FC}(S(q+p))$ . The  $\mathcal{B}_k^*$  for  $\text{FC}(S(q))$  are each loaded to  $\mathcal{C}_{\text{PR}}^{R_{\text{FC}}(S(q))}$ ,  $\mathcal{C}_{\text{PR}}^{L_{\text{FC}}(S(q))}$ , and  $\mathcal{C}_{\text{PR}}^{T_{\text{FC}}(S(q))}$  (collectively written as  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(q))}$ ), and  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(p))}$  for  $\text{FC}(S(p))$ . The corresponding 2AALUs are activated for executing  $\text{FC}(S(q)) + \text{FC}(S(p))$ , after which  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(q+p))}$  is loaded.

$\Xi_{\text{PR}}$  is contained within the 2AALU cluster  $\Xi_{2\text{AALU}} := \left( \bigcup_{\ell=2}^{\mathcal{L}-1} \mathcal{A}_{\ell,j} \right) \cup \mathcal{M}_{2\text{AALU}} \cup \Xi_{\text{PR}}$ , with  $\Xi_{\text{PR}}$  at the inner-most nest level  $\ell = 1$  and  $\mathcal{M}_{2\text{AALU}}$  at level  $\mathcal{L}$ .  $\Xi_{2\text{AALU}}$  performs operations on operands stored in the  $\mathcal{C}_{\text{PR}}$  according to parallelizable instructions. Given some  $\text{FC}(S(q))$ , these instructions amount to modifying the  $a_i$  in  $\text{FC}(S(q))$  in order to obtain an output  $\text{FC}(S(\mu(q)))$ , where  $\mu$  is the modification. Arithmetic begins when the CU instructs the LSU to prompt  $\mathcal{M}_{\text{PR}}$  to activate the operand-storing  $\mathcal{C}_{\text{PR}}$  (e.g.,  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(q))}$  and  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(p))}$ ) and alert  $\mathcal{M}_{2\text{AALU}}$  that these  $\mathcal{C}_{\text{PR}}$  are active.  $\mathcal{M}_{2\text{AALU}}$  is then instructed to retrieve from main memory, via the LSU, parallelizable instructions that instruct the  $\mathcal{A}_{\ell,j}$  in  $\Xi_{2\text{AALU}}$  at nest levels  $\ell \geq 2$  to modify the operand  $a_i$  in parallel, where operations on coefficients at lower  $i$  (i.e., further to the right) are performed by  $\mathcal{A}_{\ell,j}$  at lower  $\ell$  (i.e., innermost in the nest-structure) and greater  $i$  (i.e., further to the left) by  $\mathcal{A}_{\ell,j}$  at greater  $\ell$  (i.e., outermost in the nest-structure). Following the modification,  $\mathcal{M}_{2\text{AALU}}$  is instructed by the CU to alert the LSU, which prompts  $\mathcal{M}_{\text{PR}}$  to load the  $\mathcal{C}_{\text{PR}}$  addressed to the output number (e.g.,  $\mathcal{C}_{\text{PR}}^{\text{FC}(S(q+p))}$ ) via activation with  $\pi$  bits.

## 4.2 Nested-Clustered Design

In the Hensel architecture, each cluster is to be physically designed in nested form. From an engineering perspective, one can build rectangular containers for the 2AALUs that are placed within one another, as shown in Figure 7. The process registers are, in turn, distributed across the cavities that remain at the innermost level of the nested structure (see Figure 8).

A  $\Xi_{2\text{AALU}}$  of nest depth  $\mathcal{L}$  is designed so that, proceeding from the innermost  $\mathcal{A}$ -level  $\ell = 2$  to level  $\mathcal{L} - 1$  (with level 1 storing the  $\Xi_{\text{PR}}$  and level  $\mathcal{L}$  storing the  $\mathcal{M}_{2\text{AALU}}$ ), there are  $2^{\mathcal{L}-\ell}$  2AALUs  $\{\mathcal{A}_{\ell,2^{\mathcal{L}-\ell}}, \dots, \mathcal{A}_{\ell,1}\}$  at each level  $\ell$ . Each 2AALU at level  $\ell$  will contain 2 2AALUs at level  $\ell - 1$  (with the exception of the  $\mathcal{A}_{2,j}$ , which contain the  $\mathcal{C}_{\text{PR}}$ ) and will be contained alongside another 2AALU by a 2AALU at level  $\ell + 1$  (with the exception of  $\{\mathcal{A}_{\mathcal{L}-1,1}, \mathcal{A}_{\mathcal{L}-1,2}\}$ , which is contained by  $\mathcal{M}_{2\text{AALU}}$ ). Thus, it is convenient to describe the nest structure of  $\Xi_{2\text{AALU}}$  in terms set membership, where, as a shorthand,  $\mathcal{A}_{\ell,j} \equiv \{\}_{\ell,j}$ . With this shorthand, we can write the nest structure as follows, beginning at  $\ell = 2$  and moving outward by  $\ell + 1$ :

$$\{\}_{\ell+1,j} := \begin{cases} \{\{\{\}_{\ell,j}\}, \{\{\}_{\ell,j}\}\} & (\ell - 1) \mid 2 \\ \{\{\}_{\ell,j}, \{\}_{\ell,j}\} & (\ell - 1) \nmid 2 \end{cases}, \quad \{\}_{2,j} = \{\{\}, \{\}\} \quad (15)$$

## 4.3 Cluster Load-Store and Addressing

The address  $\chi \in \mathbb{Q}_2$  assigned to a given  $\mathcal{C}_{\text{PR}}$  is simply the ( $\mathcal{T}$ -form of the)  $\text{R}_{\text{FC}(\text{S}(-))}$ ,  $\text{L}_{\text{FC}(\text{S}(-))}$ , or  $\text{T}_{\text{FC}(\text{S}(-))}$  that it stores (i.e., some  $\mathcal{B}_k^*$ ). Intuitively, for a clustered approach to distributed storage, this addressing regime is most efficient, for any other addressing regime would simply require permutations of

numbers stored by the  $\mathcal{C}_{\text{PR}}$ , requiring that the architecture manage them and the stored  $\text{R}_{\text{FC}(\text{S}(-))}$ ,  $\text{L}_{\text{FC}(\text{S}(-))}$ , or  $\text{T}_{\text{FC}(\text{S}(-))}$  for each  $\mathcal{C}_{\text{PR}}$ , which is doubly cumbersome. Moreover, as a consequence, this  $\chi$ -scheme unifies address-generation and load-store. That is to say, because a  $\mathcal{C}_{\text{PR}}$ , by storing a  $\chi$ -address, also stores a  $\text{R}_{\text{FC}(\text{S}(-))}$ ,  $\text{L}_{\text{FC}(\text{S}(-))}$ , or  $\text{T}_{\text{FC}(\text{S}(-))}$  operand,  $\pi$ -bit activation is equivalent to loading and  $\chi$ -addressing is equivalent to storage. The LSU instructs the  $\mathcal{M}_{\text{PR}}$  to "load" a  $\mathcal{C}_{\text{PR}}$  by simply activating the  $\mathcal{C}_{\text{PR}}$  which stores the relevant operand as its address. Thus, generally, for a given  $n \in \mathbb{Q}_2$ , the LSU will prompt the  $\mathcal{M}_{\text{PR}}$  to activate/"load" a given  $\mathcal{C}_{\text{PR}}^{(-)\text{FC}(\text{S}(n))}$  by issuing an operational bit  $\pi$  and address  $\chi$  via the function  $\alpha$ ,

$$\alpha : (\chi, \pi) \rightarrow \mathcal{C}_{\text{PR}}^{(-)\text{FC}(\text{S}(n))} \quad (16)$$

A given  $\mathcal{C}_{\text{PR}}^{(-)\text{FC}(\text{S}(n))}$  accepts the  $\pi$  if the  $\chi$  matches its address. Once the relevant PRs are activated,  $\mathcal{M}_{\text{PR}}$  notifies the LSU and the CU notifies  $\mathcal{M}_{2\text{AALU}}$ .

As an example, suppose that the Hensel CPU receives an input for adding two numbers. Let's begin with the first, which we'll call  $q$ . First, the CU prompts the LSU to retrieve the  $\mathcal{T}$ -form of its FC 1-2025 encoding,  $\text{FC}(\text{S}(q))$ , from main memory, and then to relay it to  $\mathcal{M}_{\text{PR}}$ , which sends operational bits  $\pi$  to the various  $\mathcal{C}_{\text{PR}_i}$  whose  $\chi$ -addresses match the  $\text{R}_{\text{FC}}$ ,  $\text{L}_{\text{FC}}$ , and  $\text{T}_{\text{FC}}$ ; that is to say, it sends  $\pi$  bits and  $\chi$ -addresses to  $\mathcal{C}_{\text{PR}}^{\mathcal{B}_1^*}$ ,  $\mathcal{C}_{\text{PR}}^{\mathcal{B}_2^*}$ , and  $\mathcal{C}_{\text{PR}}^{\mathcal{B}_3^*}$  (which we can also write as  $\mathcal{C}_{\text{PR}}^{\text{R}_{\text{FC}}}$ ,  $\mathcal{C}_{\text{PR}}^{\text{L}_{\text{FC}}}$ , and  $\mathcal{C}_{\text{PR}}^{\text{T}_{\text{FC}}}$ ) in order to activate/"load" them. Following  $\Xi_{2\text{AALU}}$  computations on the operands  $\mathcal{B}_k^*$ ,  $\mathcal{M}_{\text{PR}}$  sends the outputs to the LSU, which retrieves a single  $\mathcal{T}$ -form output (i.e., triple-tree).  $\mathcal{M}_{\text{PR}}$  sends a deactivation bit  $\pi^\delta$  to the original  $\text{FC}(\text{S}(q))$ -storing  $\mathcal{C}_{\text{PR}}$  and new  $(\pi, \chi)$  to (the three  $\mathcal{B}_k^*$ -addressed)  $\text{FC}(\text{S}(\mu(q)))$ -storing  $\mathcal{C}_{\text{PR}}$  via  $\alpha$  functions.

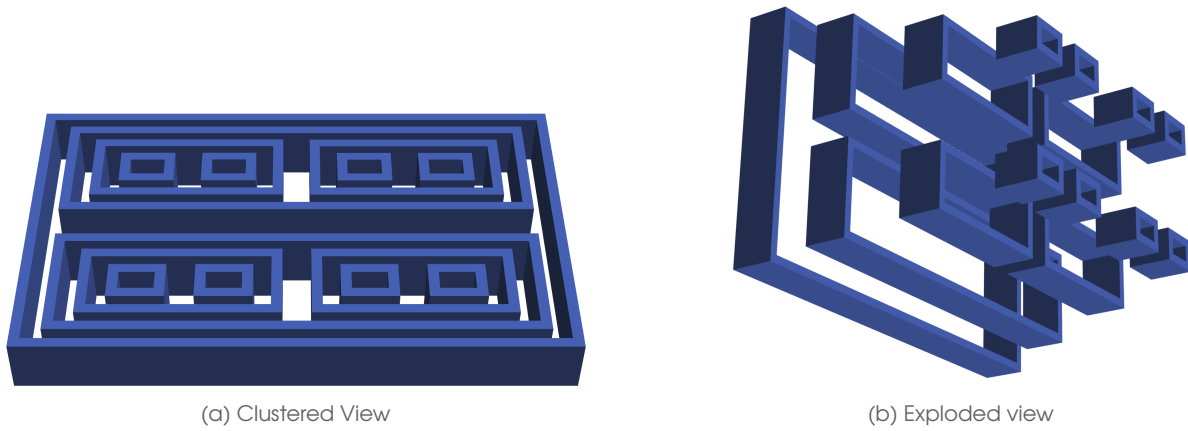


Figure 7: Illustration of the nested structure in  $\Xi_{2AALU}$ .

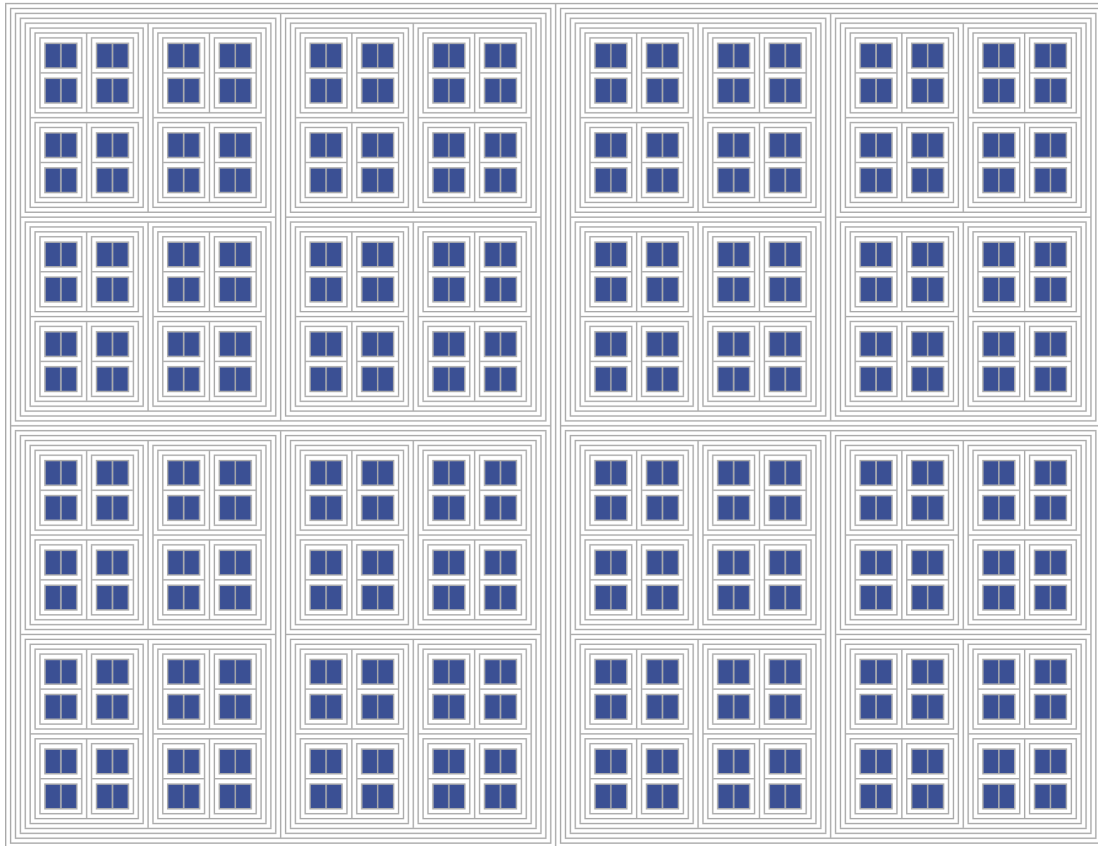


Figure 8: Cluster distribution of  $\mathcal{C}_{PR}$  at the innermost level of  $\Xi_{2AALU}$ . (Note that the number of nested layers for  $\Xi_{2AALU}$  in this example is greater than in Figure 7, simply for purposes of visual variety.)

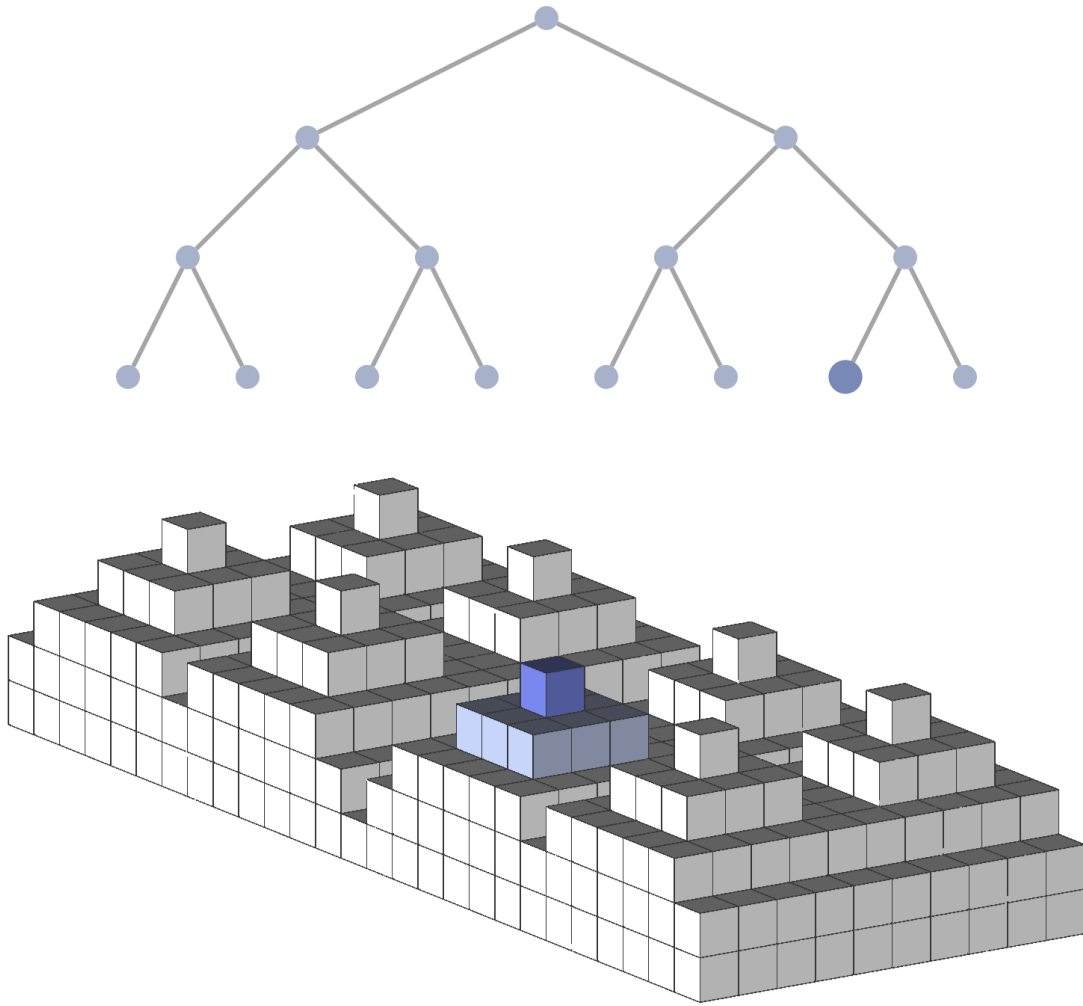


Figure 9: Illustration of the correspondence between  $\mathcal{B}_k^*$ -form tree structure and  $\mathcal{C}_{PR}$  cluster position. In this and other similar illustrations, the  $\mathcal{C}_{PR}$  are positioned at the "top", with layers "below" being 2AALU layers. Here, the top-most layers is nest-innermost in  $\Xi_{2AALU}$  and the bottom-most layer is nest-outermost in  $\Xi_{2AALU}$ .

## 5 Parallelized Arithmetic

In principle, the Hensel CPU architecture can perform arithmetic in either parallelized or sequential fashion. We will devote space in this report to the parallelized case because it is more serviceable to high-performance and accelerated computing. Parallelization is to be performed by 2AALUs simultaneously in  $\Xi_{2\text{AALU}}$  according to stored instructions, which are encoded according to FC 2-2025, the Future Computing standard for parallelized arithmetic instructions in  $\mathbb{Q}_2$ .

### 5.1 Parallel Arithmetic in $\mathbb{Q}_2$ via Coefficient-Modification

Hensel FC 2-2025 specification begins with the observation that arithmetic in  $\mathbb{Q}_2$  is but a modification of the  $a_i$  coefficients that appear in the 2-adic expansion of one 2-adic number to obtain the coefficients in the 2-adic expansion of another. Given  $\mathcal{T}$ -form FC-1-2025-encoded operands, and with respect to the  $\Xi_{\text{PR}}$  cluster, this amounts to modification of  $\chi$ -addresses. That is to say, rather than modify all (infinitely many)  $a_i$ , one modifies the entries in  $\mathcal{B}_1^*$ ,  $\mathcal{B}_2^*$ , and  $\mathcal{B}_3^*$ , which are each stored in particular  $\mathcal{C}_{\text{PR}}$  in the cluster. Thus, from a coding-theoretic perspective, whereas 2-adic arithmetic is typically an infinite  $a_i$ -modification process, here it is a (finite)  $\chi$ -modification process: arithmetic takes a  $\mathcal{T}$ -form  $\text{FC}(\mathcal{S})(q)$  stored in  $\mathcal{C}_{\text{PR}}^{\mathcal{B}_k^*}$  in the cluster, performs a parallelized modification  $\mu$ , and returns an output  $\text{FC}(\mathcal{S}(\mu(q)))$  whose  $\mathcal{B}_k^*$  are stored in their respective  $\mathcal{C}_{\text{PR}}^{\mathcal{B}_k^*}$  in the cluster.

For a binary arithmetic operation  $\circ$  and operands  $\text{FC}(\mathcal{S})(q)$  and  $\text{FC}(\mathcal{S})(p)$ , the computational depth of parallelization (i.e., the number of parallelized operations) will depend on  $\text{FC}(\mathcal{S}(q \circ p))$ . When  $\mathcal{M}_{2\text{AALU}}$  is alerted that the operands are  $\text{FC}(\mathcal{S})(q)$  and  $\text{FC}(\mathcal{S})(p)$  and the operation is  $\circ$ , it retrieves the instructions for computing  $\text{FC}(\mathcal{S}(q \circ p))$  from main memory. Such instructions are encoded according to the FC 2-2025 standard.

### 5.2 FC 2-2025 Encoding

From a coding-theoretic perspective, one could view coefficient modification as performed by flipping a bit (e.g.,  $(1, 1, 1) \rightarrow (1, 1, 0)$ ), inserting a new bit (e.g.,  $(1, 1) \rightarrow (0, 1, 1)$ ), or removing a bit (e.g.,  $(0, 1, 1) \rightarrow (1, 1)$ ). However, given the distributed nature of load-store in the Hensel processor, one can think of  $a_i$  modification as  $\chi$ -modification. Thus, given some input  $\text{FC}(\mathcal{S}(q))$  and modification  $\mu$ , one can obtain (within the permissible bit-width), the output  $\text{FC}(\mathcal{S}(\mu(q)))$  in a different fashion. Rather than do so via instructions for bit flipping, insertion, and deletion, instead the Hensel processor follows instructions that give the shortest sequence of nest-hops within the cluster from the  $\mathcal{C}_{\text{PR}}$  with  $\chi$ -addresses giving  $\text{FC}(\mathcal{S}(q))$  to those with  $\chi$ -addresses giving  $\text{FC}(\mathcal{S}(\mu(q)))$ . Thus, coefficient-modification is reduced to parallelized hopping in the nested cluster between  $\mathcal{C}_{\text{PR}}$ .

The procedure is as follows. A 2AALU  $\mathcal{A}_{\ell,j}$  (for  $2 \leq \ell \leq \mathcal{L} - 1$ ) in  $\Xi_{2\text{AALU}}$  is designed such that it can accept an input (a 2AALU token  $\tau$ ) from some  $\mathcal{A}_{\ell,j'}$  (i.e., at the same level) or send a  $\tau$  to some  $\mathcal{A}_{\ell,j'}$ .  $\Xi_{2\text{AALU}}$  uses this capability to perform FC-2-2025 encoded hops, as discussed below. Given that particular 2AALUs enclose particular  $\mathcal{C}_{\text{PR}}$  at level  $\ell = 1$ , performing such hops in parallel amounts to a  $\chi$ -modification such that, given the  $\mathcal{B}_k^*$  for some  $\text{FC}(\mathcal{S}(q))$  and  $\mu = (\circ, p)$ ,  $\Xi_{2\text{AALU}}$  can efficiently locate the  $\mathcal{B}_k^*$  for  $\text{FC}(\mathcal{S}(\mu(q)))$  (where a computation that takes  $\mathcal{B}_k^*$  as input and gives new  $\mathcal{B}_k^*$  as output is a  $\chi$ -modification by definition).

There are two ways in which a given 2AALU can pass a  $\tau$ . Because each 2AALU at level  $\ell + 1$  contains two 2AALUs at level  $\ell$  (i.e.,  $\{\mathcal{A}_{\ell,1}, \mathcal{A}_{\ell,11}\}$ ), then  $\tau$  can be passed according to one of two shifting hops:

$$\tau h_{\ell}^{\sigma} : \mathcal{A}_{\ell,1} \rightarrow \mathcal{A}_{\ell,11} \quad (17)$$

or

$$\tau h_{\ell}^{\bar{\sigma}} : \mathcal{A}_{\ell,11} \rightarrow \mathcal{A}_{\ell,1} \quad (18)$$

The absence of an operation is a non-hop  $\tau h_{\ell}^{\perp}$ .

The standard encoding for hop instructions, which will be known as FC 2-2025, is as follows:

$$(\perp, \mathfrak{P}_R, \perp, \mathfrak{P}_L, \perp, \mathfrak{P}_T) \quad (19)$$

where  $\mathfrak{P}_R$  is a sublist of instructions for  $R_{FC}$ ;  $\mathfrak{P}_L$ , for  $L_{FC}$ ; and  $\mathfrak{P}_T$ , for  $T_{FC}$ . The elements populating these  $\mathfrak{P}_{(-)}$  sublists are  $\tau h_\ell^\sigma$  and  $\tau h_\ell^{\bar{\sigma}}$  (as well as  $\tau h_\ell^\perp$ ) where  $\tau h_\ell^\sigma$  is encoded as  $(0, 1)$  and  $\tau h_\ell^{\bar{\sigma}}$  is encoded as  $(1, 0)$ . (Additionally, a non-hop is encoded as  $(0, 0)$ .) Each instruction is in turn separated by a  $\perp$ .

FC 2-2025 encoding can be applied to instructions for addition, subtraction, multiplication, or division operations whose inputs and outputs are  $n \in \mathbb{Q}_2$  within the allowed bit-width, with the advantage of forgoing operations such as carrying in carry arithmetic; the 2AALUs just perform modifications according to stored instructions. For instance, recall that the FC 1-2025 encoding for  $\frac{1}{3}$  is  $(\perp, 0, 1, \perp, 1, \perp)$ . Adding  $\frac{1}{5}$  and  $\frac{1}{3}$  is encoded in the following FC 2-2025 instruction:

$$(\perp, \perp, (1, 0), \perp, (0, 1), \perp, (0, 1), \perp, (0, 0), \perp, \perp, (0, 1), \perp, (1, 0), \perp, (1, 0), \perp, (0, 1), \perp, \perp) \quad (20)$$

To be more precise, this instruction performs a modification on  $FC(S(\frac{1}{3}))$  and returns  $FC(S(\mu^{(+, \frac{1}{5})}(\frac{1}{3}))) = FC(S(\frac{8}{15}))$ .

However, for purposes of evaluating parallelization performance, it is better to write FC-2-2025-encoded instructions in  $\mathfrak{P}$ -form, where the  $\mathfrak{P}$ -form of a given  $\mathfrak{P}_{(-)}$  instruction list for a modification  $\mu(FC(S(q)))$  is no more than a list  $\mathcal{H}$  of  $\tau h_\ell^{(-)}$  instructions, which are written from right to left for 2AALUs from levels  $\ell = 2$  to  $\ell = \mathcal{L} - 1$ :

$$\begin{aligned} \mathfrak{P}(\mu(FC(S(q)))) &:= \mathcal{H} \\ &= (\tau h_m^{(-)}, \dots, \tau h_2^{(-)}) \end{aligned} \quad (21)$$

where  $\mathcal{L} - 1 \geq m$ . Thus, a parallel computation performing a modification  $\mu$  (written  $\mathfrak{P}(\Xi(\mu(-)))$ ) of depth  $\mathfrak{D} = \text{Length}(\mathcal{H}) = m - 1$ , will execute  $(m - 1)$ -many  $\tau h_\ell^{(-)}$  operations. When  $\chi$ -modification

is performed for  $R_{FC}$ ,  $L_{FC}$ , or  $T_{FC}$ , each is treated as a separate number and, each having its own  $\mathfrak{P}_{(-)}$ , has its own  $\mathfrak{P}$ -form, with  $\chi$  entries beginning at  $a_0$  and operations beginning at  $\tau h_2^{(-)}$ .

### 5.3 Parallelization and Non-Archimedean Distance

Intuitively, given the nested structure of  $\Xi_{2AALU}$ , a hop operation affects the 2AALUs (and ultimately, at level  $\ell = 1$ , PRs) one can access more so, relative to  $\ell$ , when performed at outermore levels  $\ell + k$  (where  $1 \leq k \leq \mathcal{L} - 1 - \ell$ ) than at innermore levels; for instance, hop operations  $\tau h_{\ell-k}^-$  can be performed all the while remaining nested within the same 2AALU at level  $\ell$ . Thus, one might suppose that the more outer in the cluster an operation is performed (i.e., the greater  $\ell$  is), the greater its effect on the output, which is rather inefficient. However, recalling the 2-adic norm  $|\cdot|_2$ , one finds the opposite to be the case: the lower- $j$ , innermost operations have the greatest affect on the output.

This is a consequence of triple-tree data-structure and nested-clustered design, such that, for each  $\mathfrak{P}$ -form instruction, innermost 2AALU hops span a greater 2-adic distance over the PRs than outermost 2AALU hops, such that operations at innermost levels cover more distance between operand and output than those at outermost levels. Thus, it is appropriate here to discuss further the relationship between the nest structure of  $\Xi_{2AALU}$  and non-archimedean distance, particularly 2-adic distance.

Recall the 2-adic distance between  $p, q \in \mathbb{Q}_2$ :

$$|q - p|_2 = 2^{-v_2(q-p)} \quad (22)$$

where  $v_2$  is the 2-adic valuation. Given an operand  $q$  and output  $\mu(q)$ , we'll denote the 2-adic output-operand distance as follows:

$$\Delta_2^\mu(q) := |\mu(q) - q|_2 \quad (23)$$

Let's consider, in the parallelized case, the relationship between  $\Delta_2^\mathcal{H}(-) = \sum_{\ell=2}^{\mathcal{L}-1} \Delta_2^{\tau h_\ell^{(-)}}(-)$



and  $\mathfrak{D}(\mathcal{H})$ , the parallelization depth. Notably, greater  $\mathfrak{D}(\mathcal{H})$ , requiring  $\tau h_\ell^{(-)}$  at ever-greater  $\ell$ , probes smaller 2-adic distances per  $\tau h_\ell^{(-)}$  operation, or to be more precise:

$$\max \left( \Delta_2^{\tau h_\ell^{(-)}} (\text{FC}(S(q))) \right) \propto \frac{1}{\mathfrak{D}(\mathcal{H})} \quad (24)$$

Thus, for greater  $\mathfrak{D}(\mathcal{H})$ , the upper bound on  $\Delta_2^{\tau h_\ell^{(-)}}$  decreases with each additional level.

For instance, modification of  $a_0$  has the greatest effect on  $\Delta_2^\mu$ . Consider the case of  $q = 1$  and  $\mu := (+, 1)$ . In this case,  $\Delta_2^{(+,1)}(1) = 2^0 = 1$ . This only involves modification of  $a_0$  (thus,  $\mathfrak{D} = 1$ ). Or, consider the case of  $a_1$ -modification: take the case where  $q = 1$  and  $\mu := (+, 2)$ . In this case,  $\Delta_2^{(+,2)}(1) = 2^{-1} = \frac{1}{2}$ , a smaller distance. Of course, modification of  $a_0$  can affect smaller

distances too. For instance, if  $q = 32$  and  $\mu := (+, 1)$ , then  $\Delta_2^{(+,32)}(1) = 2^{-5}$ . However, there is no modification of  $a_{i>1}$  that gives a distance  $\Delta_2^{\tau h_\ell^{(-)}} \geq 2^0$  (see Appendix).

At first glance, the above statement may seem surprising, for, in the usual case of a modification, one can indeed obtain values  $\Delta_2^\mu > 1$  when negative exponents appear in the 2-adic expansion of the operand or output, that is, coefficients to the right of the "decimal point". However,  $\mathfrak{P}$ -form encodings give instructions for  $\mathcal{T}$ -form FC-1-2025 operands, which are of  $\mathcal{B}_k^*$  structure.  $T_{FC}$  entries, which do correspond to negative exponents in a typical 2-adic expansion, are encoded in  $\mathcal{B}_3^*$  beginning with  $a_0$  and thus non-negative.  $\mathcal{T}$ -form abolishes index-negativity and thereby bounds output-operand distance to  $0 \leq \Delta_2^\mu \leq 1$  (see Appendix).

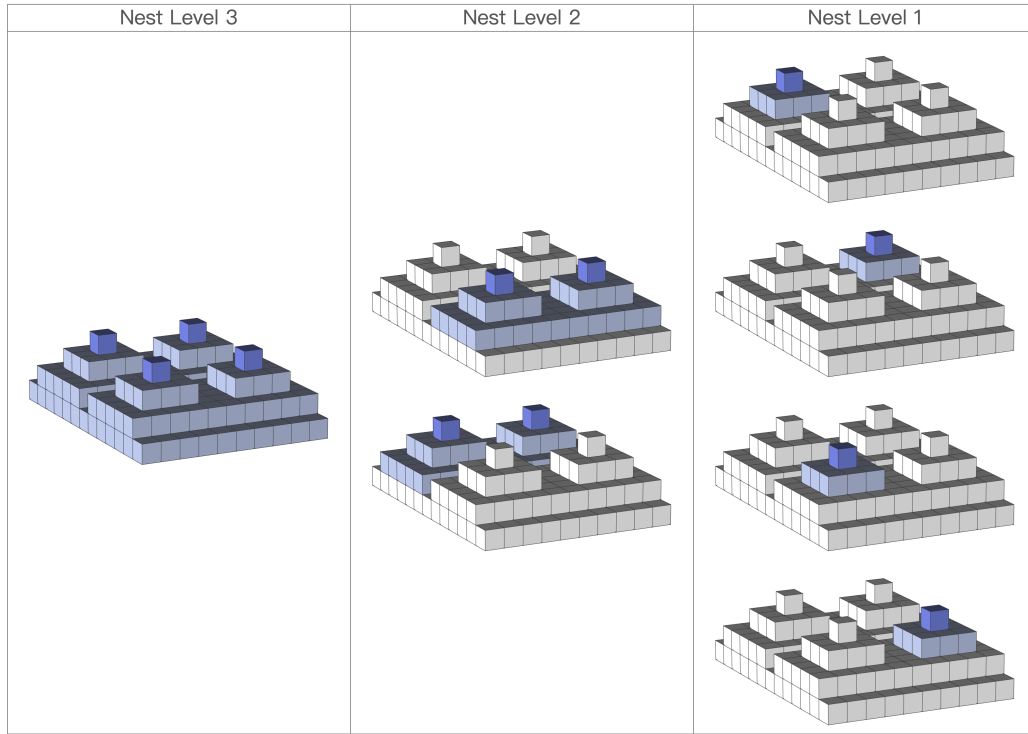


Figure 10: Table highlighting, per level, the 2AALUs a hop can reach (in light blue) and the PRs they can reach (in dark blue).

#### 5.4 $\text{Max}(\Delta_2^{\tau_{h_\ell}})$ Parallelization

If we look at  $\Delta_2^{\tau_{h_\ell}}$  per level  $\ell$ , i.e., per  $\Delta_2^{\tau_{h_\ell}^{(-)}}$ , we find that the inner-to-outer level execution of right-to-left-indexed  $\mathfrak{P}$ -form instructions necessarily maximizes, relative to depth  $\mathfrak{D}$ , the distance  $\Delta_2^{\tau_{h_2}^{(-)}}$  that can be cleared per level  $\ell$ , as shown in Figure 11. This owes to the encoding of FC-1-2025 operands in  $\mathcal{B}_k^*$ , as discussed previously, as well as the design of  $\Xi_{2\text{AALU}}$ , where right-to-left indexation corresponds with nest level  $\ell$ , such that, during right-to-left execution of  $\mathfrak{P}$ -form instructions, the 2AALU on the innermost (non- $\Xi_{\text{PR}}$ ) level (i.e.,  $\ell = 2$ ) executes the first instruction (i.e., modifies the first, right-most  $\mathcal{B}_k^*$  entry); the second  $\mathcal{B}_k^*$  entry is modified at the second-innermost level (i.e.,  $\ell = 3$ ); and so on (in parallel). As a consequence, given some

$\mathcal{A}_{2,i}$  performing a  $\tau_{h_2}^{(-)}$  hop,  $\max(\Delta_2^{\tau_{h_2}^{(-)}})$  is greater than  $\max(\Delta_2^{\tau_{h_i}^{(-)}})$ . (See the Appendix for further discussion.)

Such is advantageous for parallelization, as it means that  $\mathcal{A}_{\ell,j}$  are maximally level-efficient, per  $\ell$  (and with respect to  $\mathfrak{D}$ ), in bringing the operand to the output value. Thus, the  $\mathfrak{D}$  values for parallel computation can readily be minimized by following the principle of  $\mathfrak{D}$ -minimization through  $\Delta_2^{\tau_{h_\ell}^{(-)}}$ -maximization, where  $\mathfrak{D}$ -minimization necessarily minimizes the number of 2AALUs involved, and is thus a benchmark for parallelization efficiency. The relationship between nest level execution of FC 2-2025-encoded  $\mathfrak{P}$  instructions is illustrated, for some rudimentary cases, in Figure 12.

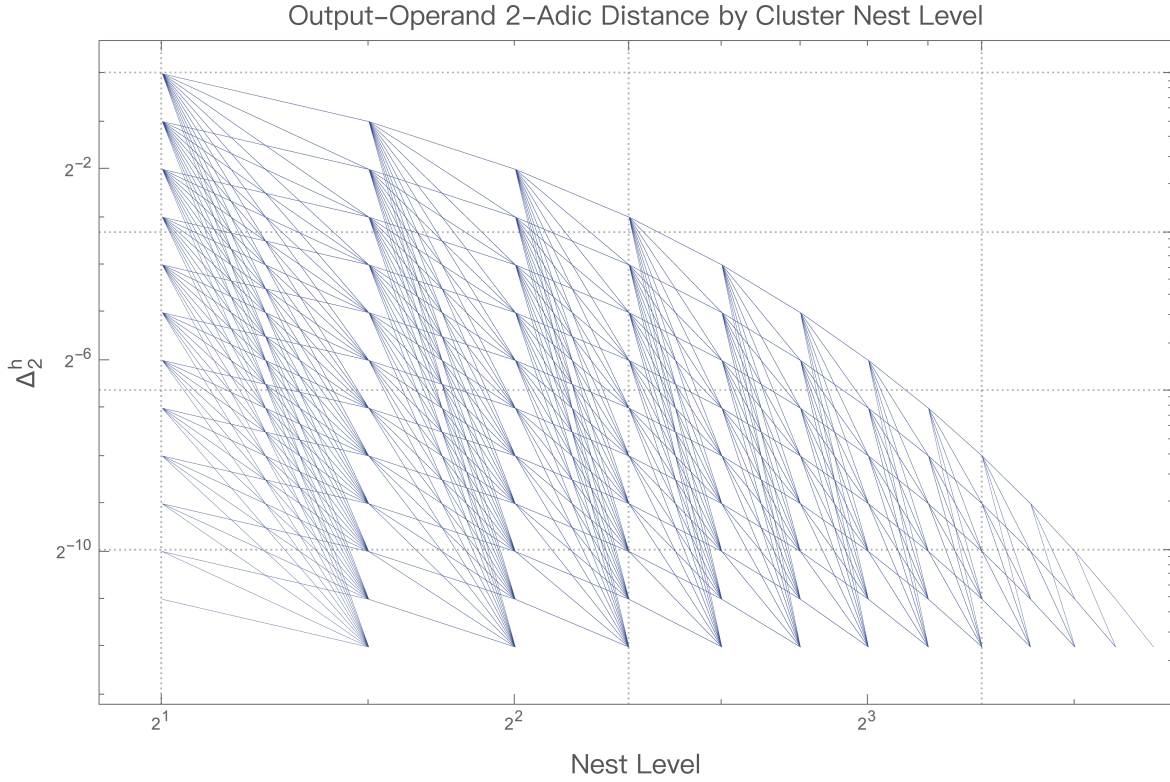


Figure 11: Plot showing possible  $\Delta_2^{\tau_{h_\ell}^{(-)}}$  values per  $\ell$  level, where  $2 \leq \ell \leq 14$ .




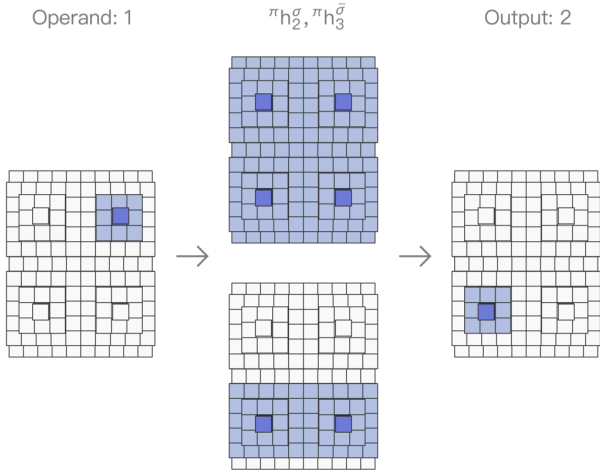

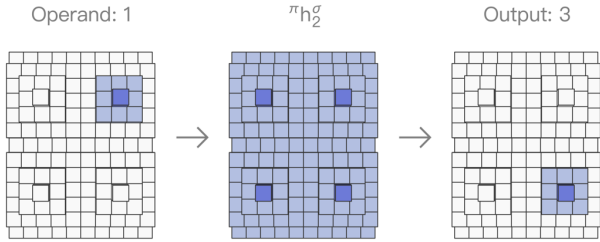

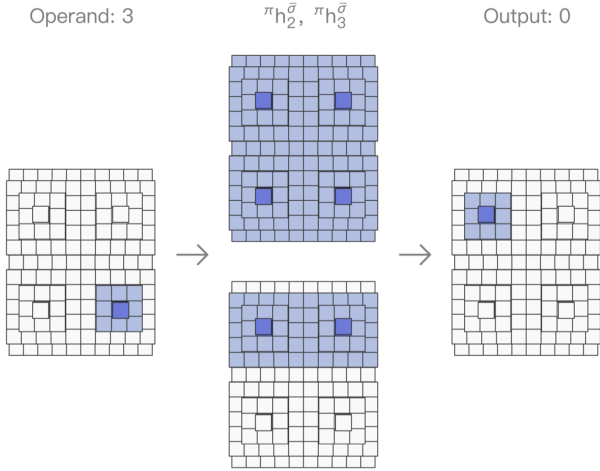
Sum	FC 2-2025 Instructions	2AALU Operations
1 + 1		<div> <div>Operand: 1</div>  <div>Output: 2</div> </div>
1 + 2		<div> <div>Operand: 1</div>  <div>Output: 3</div> </div>
3 - 3		<div> <div>Operand: 3</div>  <div>Output: 0</div> </div>

Figure 12: Elementary visualizations of FC-2-2025-encoded instructions.


Figure 13: A  $\mathfrak{P}$ -form "instruction table", where the columns are  $\mathcal{C}_{PR}$  whose  $\chi$ -addresses are the  $\mathcal{T}$ -form FC 1-2025 operand inputs  $\{0, 1, 2, 3, 4, 5, 6, 7\} \in \mathbb{Q}_2$ , and the rows are the same  $\mathcal{C}_{PR}$  whose  $\chi$ -addresses are taken as outputs. Shown in the table are the  $\tau_{h_\ell}^{(-)}$  operations in the instructions that return each output from each input.

## 5.5 Constraints on Optimization

Although parallelization lends the efficiency of simultaneity to computation, and the  $\max(\Delta_2^\mu)$  property makes operations at each nest level economical, there must necessarily exist constraints on parallelization optimality, which should be articulated so as to measure performance relative thereto. The key constraint, of course, is a depth constraint: certain computations will require many parallel operations. Depth costs are incurred when operands and outputs are many hops removed from one another in  $\Xi_{PR}$ , which will ineluctably require parallelized computation involving several 2AALUs to effectuate hops.

As an example, let's consider possible values of  $q$ , namely  $\{0, 1, 2, 3, 4, 5, 6, 7\} \in \mathbb{Q}_2$ . Let's also consider 8 values for  $\mu(q)$ , which will also be  $\{0, 1, 2, 3, 4, 5, 6, 7\}$ . Figure 13 provides a table showing the  $\mathcal{H}$  needed to obtain a given  $\mathcal{T}$ -form of  $\mathfrak{P}(\Xi(\text{FC}(S(\mu(q)))))$  from the  $\mathcal{T}$ -form of  $\text{FC}(S(q))$ . The following  $8 \times 8$  matrix

gives the  $\mathfrak{D}(\mathcal{H})$  values for each:

$$M = \begin{pmatrix} 0 & 1 & 1 & 2 & 1 & 2 & 2 & 3 \\ 1 & 0 & 2 & 1 & 2 & 1 & 3 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 1 & 2 \\ 2 & 1 & 1 & 0 & 3 & 2 & 2 & 1 \\ 1 & 2 & 2 & 3 & 0 & 1 & 1 & 2 \\ 2 & 1 & 3 & 2 & 1 & 0 & 2 & 1 \\ 2 & 3 & 1 & 2 & 1 & 2 & 0 & 1 \\ 3 & 2 & 2 & 1 & 2 & 1 & 1 & 0 \end{pmatrix} \quad (25)$$

Note that entries in  $M$  near the left-to-right diagonal are small, whereas the opposite is true for the right-to-left diagonal; in the latter case, the operands and outputs are stored in  $\mathcal{C}_{PR}$  that are hop-distal from one another in  $\Xi_{2AALU}$ . The values near these diagonals are sufficiently disparate such that if one interpolates a surface  $\mathfrak{X}$  from the matrix (i.e., with coordinates  $(i = q, j = \mu(q), M_{i,j})$ ), the gap between high- and low- $\mathfrak{D}(\mathcal{H})$  along the diagonals gives a hole in the surface (see Figure 14). Heuristically, one can think of this "topological invariant" in the interpolated surface as exhibiting the difference in  $\mathfrak{D}(\mathcal{H})$  between hop-proximal and hop-distal  $\mathcal{C}_{PR}$ .

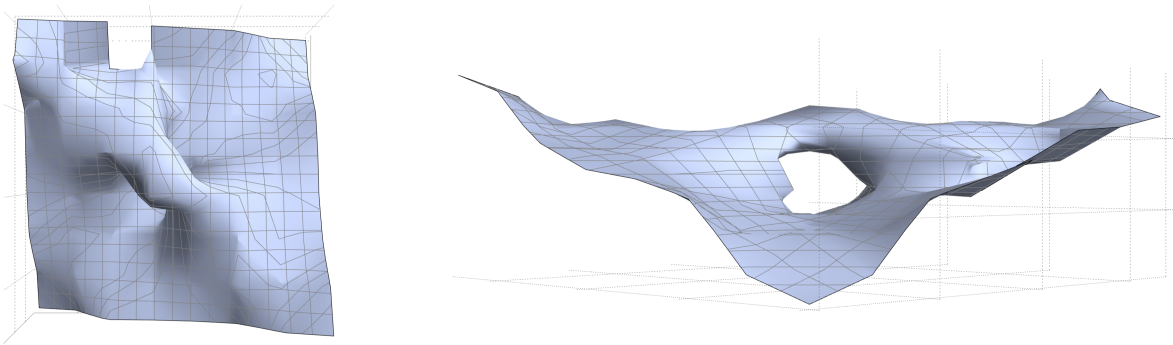


Figure 14: A 3D plot of  $\mathfrak{X}$  seen from two viewing angles.

## 6 Further Prospects

### 6.1 Error Correction and Fault Tolerance

With the prospective advantage of Hensel CPU architecture being exact arithmetic, a design that ensures checks against computational error, and is resilient to computational faults, is of paramount importance in delivering arithmetic performance. Sketched cursorily are some Hensel-architecture-compatible mechanisms for delivering error correction and fault tolerance. These sketches are all inefficient but give indications of the kind of mechanisms that can be developed.

#### 6.1.1 Error-Checking with $(\chi, \tau)$ -Payloads

Error detection for FC-1-2025-form operands can be performed with the  $\chi$ -address system. Suppose a given  $B_k^*$  is mis-transmitted (e.g., between a  $\mathcal{C}_{PR}$  and an  $\mathcal{A}$ , between the  $\mathcal{A}$  and  $\mathcal{M}_{PR}$ , etc.). Because the process begins by sending the appropriate  $\chi$  from  $\mathcal{M}_{PR}$  to the appropriate  $\mathcal{C}_{PR}$ , one could readily implement a check for operand-transmission consistency during  $\Xi_{2AALU}$  operations by re-

quiring that the 2AALUs involved continue to transmit  $\chi$  in their payloads by requiring that  $\tau$  payloads become  $(\chi, \tau)$ -payloads, in which case FC-1-2025-form errors could be easily detected and located.

#### 6.1.2 $\Delta_2^\mu$ Trajectory-Defect Checks

Because 2AALUs at a given nest level  $\ell$  can perform computations that affect  $\Delta_2^\mu$  by a certain 2-adic distance, an error in computation or transmission at any given level  $\ell$  will be evident if it effectuates a change in  $\mu(q)$  beyond the  $\Delta_2^\mu$  range permissible for that level  $\ell$ . Figure 15 gives an illustration.

#### 6.1.3 $\Xi_{2AALU}$ Consensus

One could also, albeit at the cost of efficiency, implement a distributed-consensus framework within the processor itself, namely by employing an odd number of redundant  $\Xi_{2AALU}$  clusters whose outputs are sent to  $\mathcal{M}_{PR}$  upon clearing a vote. Note that such a process does not jeopardize parallelization, for the redundant  $\Xi_{2AALU}$  could compute in parallel with respect to one another. Nonetheless, an onus would be placed on the CU, main memory, and on the  $\mathcal{M}_{PR}$ .

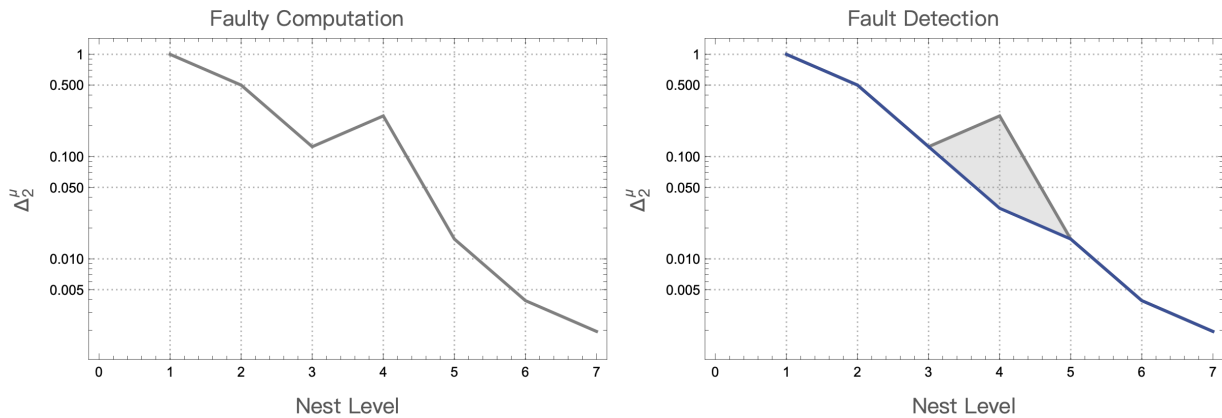


Figure 15: Illustration of  $\Delta_2^\mu$  trajectory-defect detection.

## 6.2 Supercomputing

The nested-clustered  $\Xi_{2\text{AALU}}$  is designed so that it can implement multiple F-2-2025 encoded  $\mathcal{H}$  instruction lists simultaneously. That is to say, the  $\mathcal{H}$  operations are executed in parallel and, what is more, multiple  $\mathcal{H}$  instruction lists can themselves be executed in parallel. Given an instruction superlist

$$\Pi := (\mathcal{H}_i) \quad (26)$$

the upper bound on  $\text{Length}(\Pi)$ , the number of  $\mathcal{H}_i$  that can be executed simultaneously, is the number of 2AALUs in the cluster (since, in the most efficient case, each 2AALU is performing an operation at any given time). Thus,  $\max(\text{Length}(\Pi)) = \text{Card}(\{\mathcal{A}_{\ell,j}\})$ , where

$$\text{Card}(\{\mathcal{A}_{\ell,j}\}) = \sum_{\ell=2}^{\mathcal{L}-1} 2^{\ell-1} \quad (27)$$

Assuming a highly modest 2AALU computational performance of 10 operations per second, the number of two-adic operations per second (TOPS; *pace* floating-point operations per second (FLOPS)) is

$$\max(\text{TOPS}) = 10 \sum_{\ell=2}^{\mathcal{L}-1} 2^{\ell-1} \quad (28)$$

*Ceteris paribus*, a Hensel CPU is expected to require a  $\Xi_{2\text{AALU}}$  level depth of  $\mathcal{L} \approx 57 + 2$  (where the 2 accounts for  $\ell = 1$  and  $\ell = \mathcal{L}$ ) to reach an exaTOPS performance threshold and  $\mathcal{L} \approx 67 + 2$  to cross a zetaTOPS performance threshold, as shown in Figure 17.



Figure 16: A 2AALU cluster of level depth  $\mathcal{L} = 13$ .

Comparing a nest depth of  $\mathfrak{L} \approx 67 + 2$  with the illustration in Figure 16, which shows an example of  $\mathfrak{L} \approx 13$ , one can appreciate that  $67 + 2$  is rather deep. Nonetheless, in view of the current paradigm of supercomputing, this may still be economical. For instance, Fugaku (富岳) uses 432 racks, amounting to about 7500 CPUs. By contrast, a single Hensel CPU with a single  $\Xi_{2\text{AALU}}$  of nest depth  $\mathfrak{L} \approx 65$  may still be economical (and then one wonders what one could do by applying a rack/multi-CPU approach to the Hensel case). A key comparative determinant, on

the engineering side, is the physical economy with which the nested-clustered design of the Hensel CPU can be engineered. If the nested layers can be tightly packed, Hensel may be able to deliver both arithmetic exactness as a 2-adic computer and an economical hardware approach to supercomputing. If, on the other hand, a processor with a 2AALU cluster of depth  $\mathfrak{L} \approx 67 + 2$  must be physically large, then Hensel's key comparative advantage would pertain, perhaps, to exactness alone.

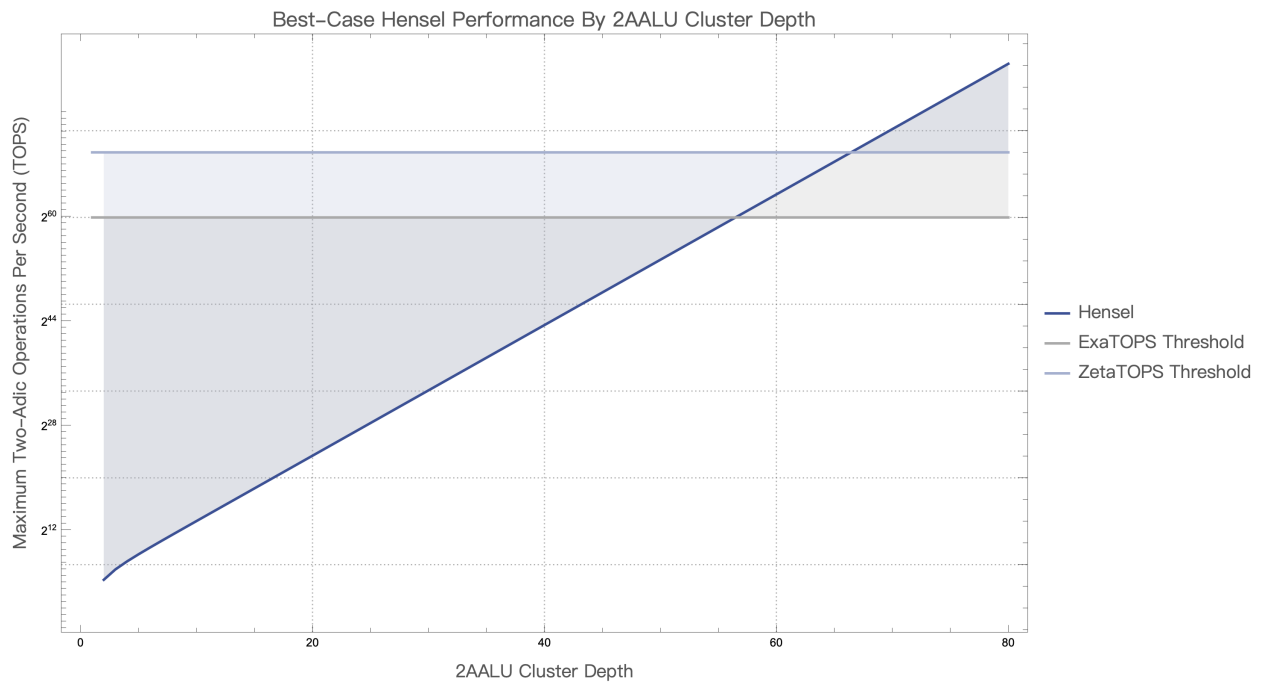


Figure 17: Best-case prediction of  $\Xi_{2\text{AALU}}$  level depth  $\mathfrak{L}$  (minus 2) required for exaTOPS and zetaTOPS performance.



## 7 Appendix: Proofs of Hensel Performance Properties

**Theorem 1.** *For every  $\mathfrak{P}$ -form instruction list with output-operand distance  $\Delta_2^{\mathcal{H}}$ , it is the case, for all  $2 \leq \ell \leq \mathfrak{D}(\mathcal{H})$ , that  $0 \leq \Delta_2^{\tau_{h_\ell}^{(-)}} \leq 1$ .*

*Proof.*  $\mathfrak{P}$ -form instructions are given for  $\Xi_{2AALU}$  computations on  $\mathcal{T}$ -form FC-1-2025-encoded operands, which are stored as  $\mathcal{B}_k^*$  (where  $R_{FC}$  is stored as  $\mathcal{B}_1^*$ ,  $L_{FC}$  is stored as  $\mathcal{B}_2^*$ , and  $T_{FC}$  is stored as  $\mathcal{B}_3^*$ ). For a given  $n \in \mathbb{Q}_2$ , it is the case that its 2-adic valuation  $v_2(n)$  can be negative – namely, when its 2-adic expansion includes negative exponents – such that the 2-adic norm  $|n|_2 > 1$ . In FC 1-2025, the coefficients of summands with negative exponents are encoded in  $T_{FC}$ . However, because  $\mathcal{T}$ -form encodings assign the entries in  $T_{FC}$  to their own binary tree  $\mathcal{B}_3^*$ , which begins its indexation at  $v_0$  (also written  $v_0^{\mathcal{B}_3^*}$ ), the entries in  $\mathcal{B}_3^*$ -encoded  $T_{FC}$  are indexed beginning with  $a_0$ . Thus,  $R_{FC}$ ,  $L_{FC}$ , and  $T_{FC}$ , in triple-tree format, can be encoded as 2-adic numbers to the left of the "decimal point", that is, as coefficients for summands in 2-adic expansions with non-negative exponents. In this case, the 2-adic valuation  $v_2(n)$  is non-negative because it takes the value of an exponent  $k \in \mathbb{Z} \cup \infty$  such that a rational  $n \in \mathbb{Q}$  can be written as  $n = 2^k \times \frac{b}{c}$ . If all exponents in the 2-adic expansion of  $n$  are now non-negative, then  $k$  must be non-negative. Thus, if  $v_2(n)$  is positive, then  $0 \leq p^{-v_2(n)} \leq 1$ , where  $|n|_2 = p^{-v_2(n)}$ . With 2-adic distance (and therefore  $\Delta_2^\mu$  too, by definition) thus bounded to  $0 \leq | \cdot |_2 \leq 1$ , it is the case that a hop operation  $\tau_{h_\ell}^{(-)}$  cannot clear an output-operand distance outside of the values  $0 \leq \Delta_2^{\tau_{h_\ell}^{(-)}} \leq 1$ .  $\square$

**Theorem 2.** *Comparing, at each  $\ell$ , the maximum reduction in 2-adic output-operand distance  $\Delta_2^{\mathcal{H}}(FC(S(q)))$  between  $FC(S(q))$ , an FC1-2025-encoded  $S$ -form operand, and  $FC(S(\mu(q)))$ , the output of a  $\Xi_{2AALU}$  computation performing a modification  $\mu$  on the*

*operand, where the instruction list  $\mathcal{H}$  contains operations  $(\tau_{h_m}^{(-)}, \dots, \tau_{h_2}^{(-)})$  to be executed in parallel by 2AALUs in  $\Xi_{2AALU}$ , it is the case that*

$$\max \left( \Delta_2^{\tau_{h_\ell}^{(-)}}(FC(S(q))) \right) > \max \left( \Delta_2^{\tau_{h_{\ell+k}}^{(-)}}(FC(S(q))) \right) \quad (29)$$

*for all  $k$  where  $\ell < k \leq m - \ell$ .*

*Proof.* In  $\Xi_{2AALU}$ ,  $\ell$ -level operations (for  $\ell \geq 2$ ), with domains  $\mathcal{A}_{\ell, I}$  or  $\mathcal{A}_{\ell, II}$ , will modify the  $(\ell - 1)$ -th (non- $\perp$ ) entry in  $FC(S(q))$ . With these entries being FC-1-2025,  $S$ -form encodings of 2-adic expansion coefficients, this modification can affect the final  $\Xi_{2AALU}(\mathfrak{P}(FC(S(q))))$  output by as much as  $\max \left( \Delta_2^{\tau_{h_\ell}^{(-)}}(FC(S(q))) \right) = 2^{-\ell+2}$ . (Here, the exponent is  $-\ell + 2$  since the first 2-adic expansion coefficient  $a_0$  is modified at level  $\ell = 2$ .) Proceeding to  $\ell + k$ , the same argument applies. A  $\tau_{h_{\ell+k}}^{(-)}$  operation can alter the 2-adic expansion of  $\mu(q)$  relative to  $q$  by as much as  $2^{-\ell-k+2}$ ; that is to say,  $\max \left( \Delta_2^{\tau_{h_{\ell+k}}^{(-)}}(FC(S(q))) \right) = 2^{-\ell-k+2}$ . It's trivial to see that  $2^{-\ell-k+2} < 2^{-\ell+2}$  for all  $k$  where  $\ell < k \leq m - \ell$ .  $\square$

**Theorem 3.** *Given a list  $\mathcal{H} = (\tau_{h_m}^{(-)}, \dots, \tau_{h_2}^{(-)})$  of  $\mathfrak{P}$ -form instructions for the modification  $\mu$  of  $q$ , the depth  $\mathfrak{D}$  is minimal, i.e.,  $\text{Length}(\mathcal{H}) = m - 1$  is minimal.*

*Proof.* Let  $\Delta_2^{\Xi} := \Delta_2^{\Xi_{2AALU}(\mathfrak{P}(\mu(-)))}(FC(S(q)))$  be the total 2-adic distance between the output  $\Xi_{2AALU}(\mathfrak{P}(FC(S(q))))$  and input  $FC(S(q))$ . On the execution side, let  $\Delta_2^{\mathcal{H}}(FC(S(q))) = \sum_{\ell}^{\mathcal{L}-1} \Delta_2^{\tau_{h_\ell}^{(-)}}(FC(S(q)))$  be the total output-operand distance cleared by hop operations at each  $\Xi_{2AALU}$  level  $\ell$ . At each level  $\ell$  in  $\Xi_{2AALU}$ , a  $\tau_{h_\ell}^{(-)}$  operation can decrease  $\Delta_2^{\Xi}$  to  $\Delta_2^{\Xi} - \Delta_2^{\tau_{h_\ell}^{(-)}}$ . Due to Theorem 1, no  $\tau_{h_{\ell+k}}^{(-)}$  operation can decrease  $\Delta_2^{\Xi}$

more than a  $\tau_{h_\ell}^{(-)}$  operation. Thus, given  $0 = \Delta_2^{\Xi} - \sum_{i=0}^n 2^{-i}$ , at a given level  $\ell$ , a  $\tau_{h_\ell}^{(-)}$  operation can reduce  $\Delta_2^{\Xi}$  by as much as  $(\sum_{i=0}^n 2^{-i}) - 2^{-\ell+2} = \max\left(\Delta_2^{\Xi} - \Delta_2^{\tau_{h_\ell}^{(-)}}\right)$ , and, with operations in  $\mathcal{H}$  instructions being encoded by  $\tau_{h_\ell}^{(-)}$ , there is no smaller level-respecting list that reduces  $\sum_{i=0}^n 2^{-i}$  to zero using  $\Delta_2^{\tau_{h_\ell}^{(-)}}$  operations than

$$\max\left(\sum_{i=0}^n 2^i - \Delta_2^{\tau_{h_m}^{(-)}}\right), \dots, \max\left(\sum_{i=0}^n 2^i - \Delta_2^{\tau_{h_2}^{(-)}}\right) \quad (30)$$

This list is of length  $m - 1$  and sums to

$$\sum_{j=2}^m \max\left(\Delta_2^{\Xi} - \Delta_2^{\tau_{h_j}^{(-)}}\right) = \Delta_2^{\Xi} - \Delta_2^{\mathcal{H}} = 0 \quad (31)$$

Thus, because the minimal number of operations is  $\mathfrak{D}(\mathcal{H}) = m - 1$ , it is the case that Length  $(\mathcal{H})$  is minimal.  $\square$

**Theorem 4.** *For every  $q \in \mathbb{Q}_2$ , its FC 1-2025 encoding  $\text{FC}(S(q))$  is unique.*

*Proof.* For a given  $q \in \mathbb{Q}_2$ , FC 1-2025 encodes, in compressed form, the coefficients  $a_i$  in the S-form 2-adic expansion  $\sum_{i=z}^{\infty} a_i 2^i$  of  $q$ , which gives a Cauchy sequence  $(\alpha_n)_n$ . It is a [well-known theorem](#) in  $p$ -adic analysis that the Cauchy sequences and elements are unique; they uniquely satisfy the following properties:

- $0 \leq \alpha_n \leq 2^n - 1$ ,  $n \geq 1$
- $\alpha_n = \alpha_{n-1} \pmod{2^{n-1}}$
- For a given  $x \in \mathbb{Z}_2$ , there exists a unique  $\alpha_n$  such that  $|x - \alpha_n| \leq 2^{-n}$

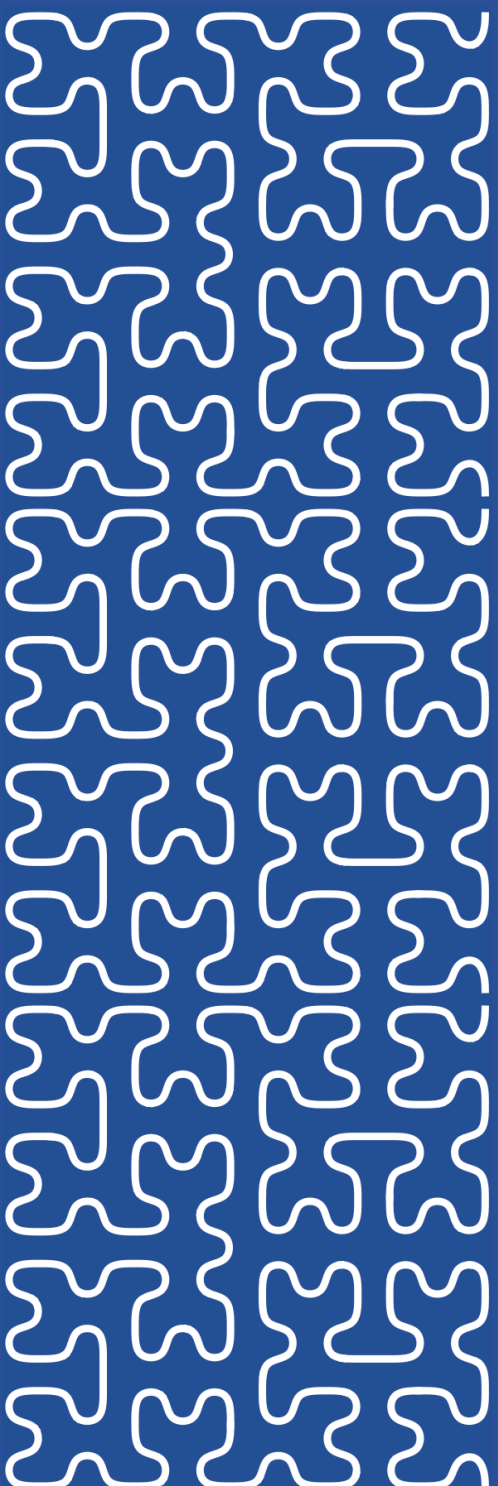
FC 1-2025 encodes  $T_{\text{FC}}$  and  $L_{\text{FC}}$ , which, being non-repeating, are thus unique due to

the uniqueness of  $a_i$ . It remains to show that the non-repeating  $a_i$ , encoded only once (i.e., without repetition) as  $R_{\text{FC}}$ , do not jeopardize uniqueness. Consider a 2-adic expansion  $E$  of some  $q \in \mathbb{Q}_2$ , whose coefficients repeat beginning at  $a_w$ , with each repetitive subsequence being of length  $L$ , such that  $R_{\text{FC}(S(E))} = (a_{w+L-1}, \dots, a_w)$ . One can ask if there could exist some  $R'_{\text{FC}(S(E))}$  such that, despite it beginning with  $a_w$  and not containing any repeating subsequences,  $R'_{\text{FC}(S(E))} \neq R_{\text{FC}(S(E))}$  (thereby showing  $R_{\text{FC}(S(E))}$  to be non-unique). This would require an instance in which  $a_{w+i} - a'_{w+i+L} \neq 0$  (where  $0 \leq i \leq L-1$ ), which would contradict the definition of  $R'_{\text{FC}(S(E))}$  as repetitive. Thus, with  $a_i$  being unique,  $R_{\text{FC}}$  are also unique. Coding-theoretic non-uniqueness can, in principle, arise whenever it is possible to insert arbitrarily many 0 coefficients (e.g., in the case of  $0 \in \mathbb{Q}_2$ ), but FC 1-2025 protects against this by forbidding it by convention. Finally, coding-theoretic collisions arising when FC 1-2025 encodings use the "same" entries (e.g., encoding  $\frac{1}{2}$  with the single coefficient 1 and encoding 1 with the single coefficient 1), are prevented via the use of  $\perp$  separators between  $T_{\text{RC}}$  and  $L_{\text{RC}}$ , which are unique for each  $n \in \mathbb{Q}_2$  due to the uniqueness of  $a_i$ .  $\square$

**Theorem 5.** *For every  $p, q \in \mathbb{Q}_2$  and binary arithmetic operation  $\circ$ , there exists an FC 2-2025 encoded instruction that takes  $q$  as its input and  $\mu(q)$  as its output where  $\mu := (\circ, p)$ .*

*Proof.* Because the FC 1 2-2025 encodings of  $p, q \in \mathbb{Q}_2$ , as well as  $p \circ q \in \mathbb{Q}_2$  are unique, there always exists an instruction list of right-to-left coefficient-modifications that takes  $q \in \mathbb{Q}_2$  as an input and gives  $p \circ q \in \mathbb{Q}_2$  as an output. Non-existence of such an instruction list would contradict the amenability of all  $n \in \mathbb{Q}_2$  to unique 2-adic expansions. Theorem 4 on the uniqueness of FC 1-2025 encodings extends this guarantee of instruction existence to FC 2-2025 instructions.  $\square$





# SciSci Research

サイサイ・リサーチ

Published by SciSci Press