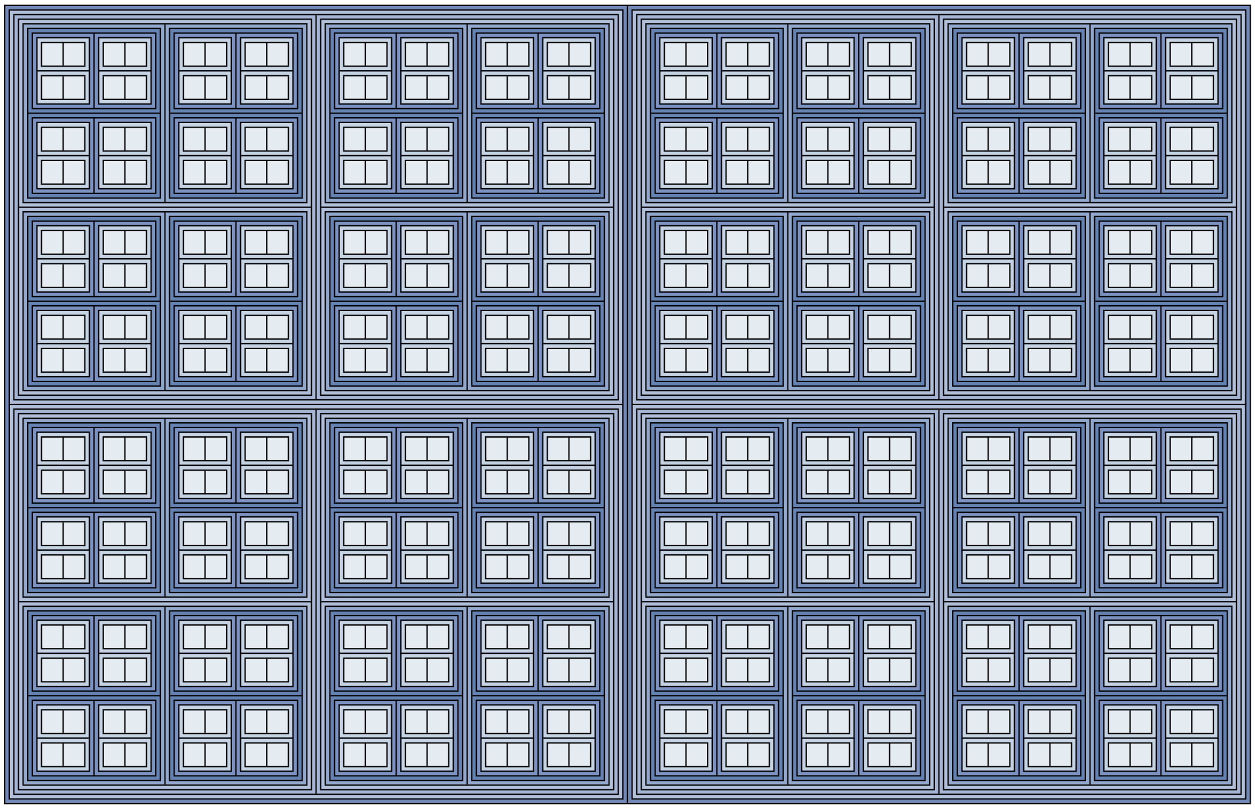
Inventions No. 1
sci–sci.org/hensel–cpu
DOI: 10.5281/zenodo.17024982

SciSci Research
サイサイ・リサーチ

Future Computing
コンピュー



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

James Douglas Boyd

SciSci Research, Inc. + Future Computing

SciSci Research, Inc.
Boulder, Colorado, United States
www.sci–sci.org

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.17024982

Contents

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 accuracy 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,

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

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

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

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

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

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.

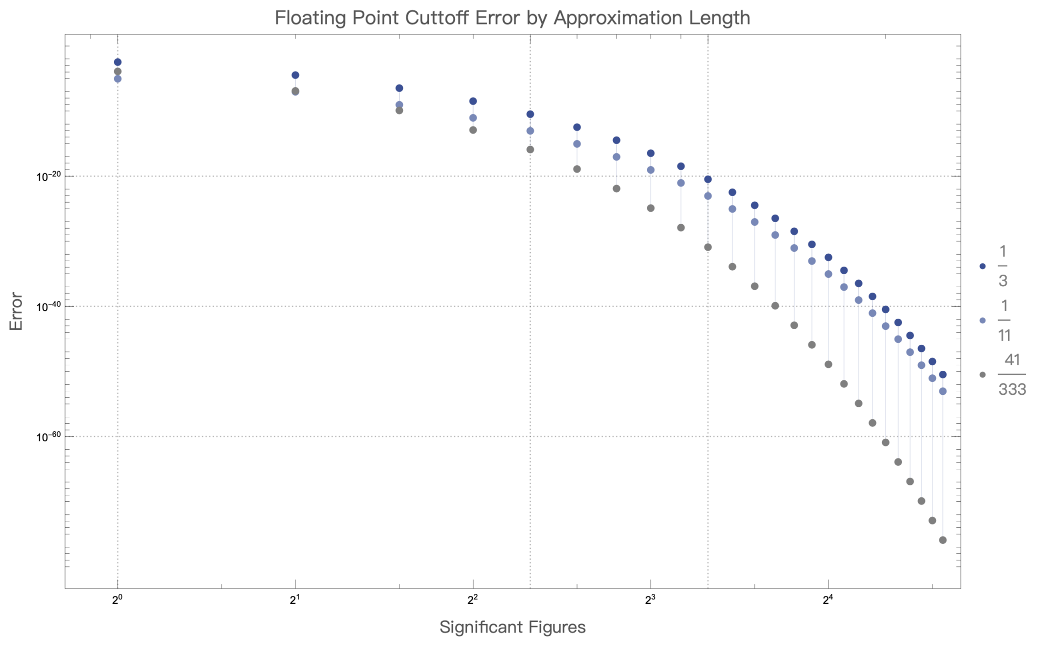


Figure 1: A plot of approximation errors for $\frac{1}{3}$ (i.e., $\frac{1}{3} - \frac{3.3\times10^{i}}{10^{i+1}}$, $i \in \{1, 3, \ldots 49\}$), $\frac{1}{11}$ (i.e., $\frac{1}{11} - \frac{\sum_{k=0}^{i} 9\times10^{k+1}}{10^{i+3}}$, $i \in \{2, 4, \ldots 50\}$), and $\frac{41}{333}$ (etc.) as one increases i.

1.2 Report Scope

With reports on the Virtual Hensel and 2AALUs now published, the role of this original report on the Hensel CPU can now be lent greater context. It is not a reference on how the computing with the Hensel architecture works. The Virtual Hensel report was written to provide and explain the Virtual Hensel as an elementary proof-of-principle demonstration of exact computation with the Hensel architecture. As a complement to the Virtual Hensel report, the 2AALU report offers a (relatively short) overview of how the capabilities exhibited by the Virtual Hensel are realized at the level of circuit design and combinational logic. Nonetheless, these two reports, with great regularity, refer back conceptually to this report. Moreover, both the Virtual Hensel and 2AALU reports, very much preoccupied with the practical details of general exact computing capability, seldom discuss matters to do with p-adic analysis or $\mathbb{Q}_2$, which are very much central to this report. Thus, this original report serves, and is expected to serve for some time, as a prerequisite text on the Hensel architecture in a manner that guides the reader from topics in pure mathematics such as 2-adic expansions, Cauchy sequences, and non-archimedean distance to features unique to the Hensel architecture such as FC encodings, 2AALUs, nested carrier packaging, χ-addresses, and distributed load-store. In many instances, this report stops short of explaining how architectural features – imbricating computer engineering strategies with number-theoretic affordances – are realized in practice as computing capabilities, instead remarking that the reader can refer to the Virtual Hensel and 2AALU reports.

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 Ξ_{2AALU}, which can perform arithmetic in parallelized fashion.

2.2.2 PR Cluster

Processor registers are also designed in a cluster, Ξ_{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 Ξ_{PR}, coordinating with the load-store unit (LSU), are instructed by the control unit (CU) to supply operands to Ξ_{PR} for parallelized execution and to receive outputs.

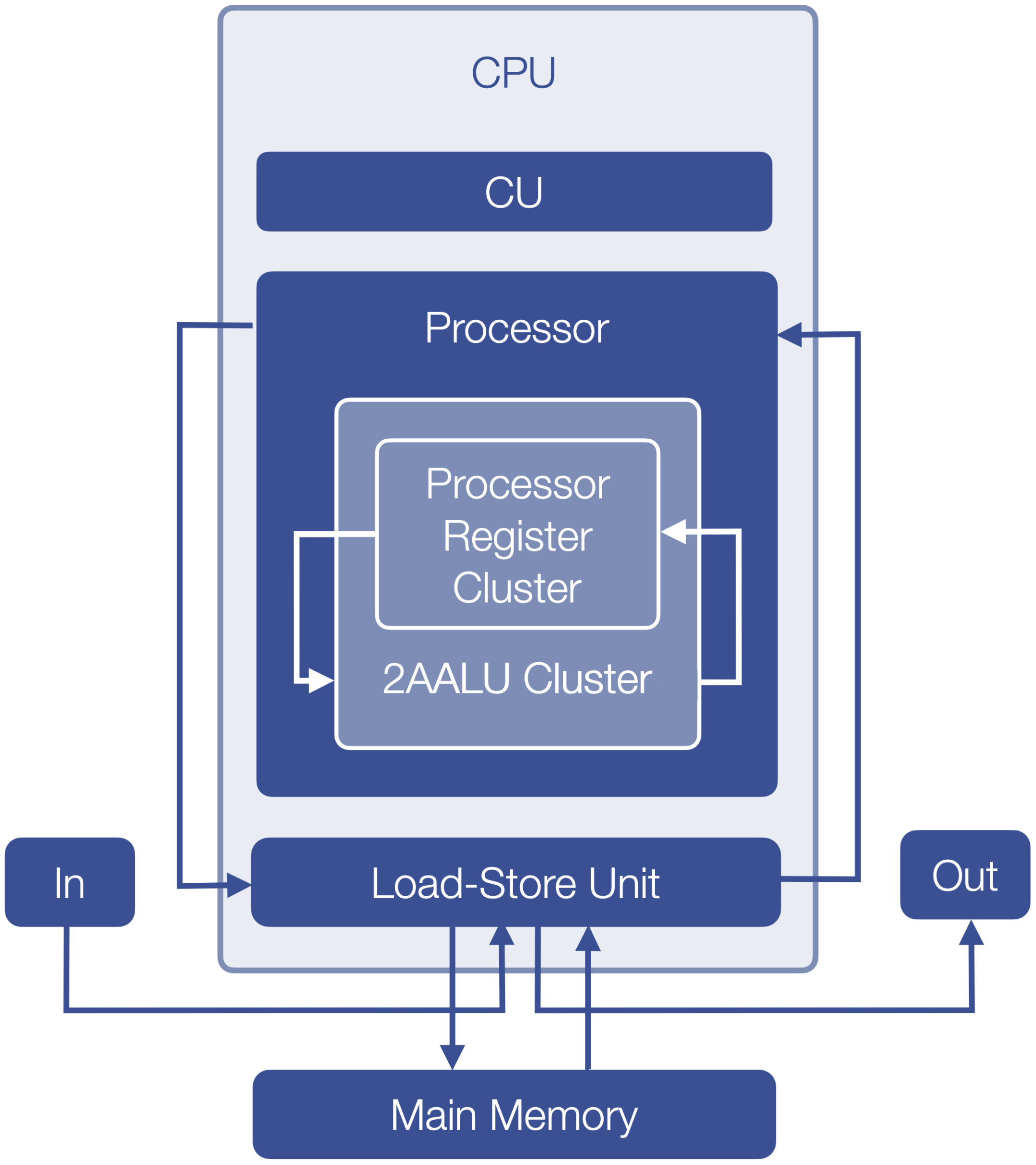


Figure 2: Block diagram of the Hensel CPU, processor, control unit (CU), load-store unit (LSU), and main memory, highlighting 2AALU and processor register cluster relations. (For simplicity of presentation, in- and outgoing arrows with respect to the the CU are omitted.)

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

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

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,

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

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

$$\begin{aligned}\frac{1}{3} &= 1 + 2 + 2^3 + 2^5 + 2^7 + 2^9 + 2^{11} \\ &+ 2^{13} + 2^{15} + 2^{17} + 2^{19} + \ldots\end{aligned} \tag{6}$$

(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,

$$\frac{1}{3} = \overline{01}1._2 \tag{7}$$

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

$$\begin{aligned}\frac{1}{5} &= 1 + 2^2 + 2^3 + 2^6 + 2^7 + 2^{10} + 2^{11} \\ &+ 2^{14} + 2^{15} + 2^{18} + 2^{19} + \ldots\end{aligned} \tag{8}$$

which can be abbreviated as

$$\frac{1}{5} = \overline{0011}01._2 \tag{9}$$

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:

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

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:

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

R_{FC} is the sublist of repeating coefficients in the 2-adic expansion (e.g., $\overline{0011}$ for $\frac{1}{5}$). Non-empty R_{FC} encodings always begin with a 1, and are of length ≥ 2. (For instance, $R_{FC}(-1) := (1,1)$.) Generally, for all sublists, sequences of 0's – e.g., for integers – are treated as empty R_{FC} encodings.) L_{FC} is the sublist of non-repeating coefficients to the left of the "decimal" point (e.g., $(0,1)$ for the case of $\frac{1}{5}$), and are minimized so as to not include repeated entries in R_{FC}, but are written so that R_{FC} can begin with a 1. 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 $.1_2$). $\perp$ is the "no operation" symbol separating R, L, and T. For instance,

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

To take another example,

$$\begin{aligned}FC\left(S\left(\frac{1}{5}\right)\right) &:= \\ &(\perp, (0,0,1,1), \perp, (0,1), \perp, ())\end{aligned} \tag{13}$$

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.

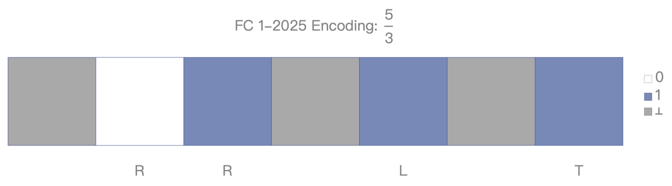


Figure 3: Array plot visualization (with color legend) of the FC 1-2025 Encoding of $\frac{5}{3}$.

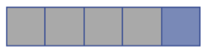
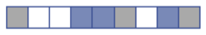
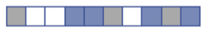
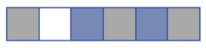
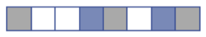
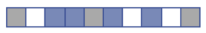
Rational	2–adic Expansion	2–adic Number	FC2 1–2025 Encoding
$\frac{1}{2}$	$2^{-1} = \frac{1}{2}$	0.1_2	
$\frac{1}{5}$	$1 + 2^2 (1 + 2^4 + 2^8 + \ldots) + 2^3 (1 + 2^4 + 2^8 + \ldots) = 1 + \frac{2^2}{1 - 2^4} + \frac{2^3}{1 - 2^4} = \frac{1}{5}$	$\overline{0011}01._2$	
$\frac{7}{10}$	$2^{-1} + 1 + 2^2 (1 + 2^4 + 2^8 + \ldots) + 2^3 (1 + 2^4 + 2^8 + \ldots) = 2^{-1} + 1 + \frac{2^2}{1 - 2^4} + \frac{2^3}{1 - 2^4} = \frac{7}{10}$	$\overline{0011}01.1_2$	
$\frac{1}{3}$	$1 + 2 (1 + 2^2 + 2^4 + \ldots) = 1 + \frac{2}{1 - 2^2} = \frac{1}{3}$	$\overline{01}1._2$	
$\frac{3}{7}$	$1 + 2^2 (1 + 2^3 + 2^6 + 2^9 + \ldots) = 1 + \frac{2^2}{1 - 2^3} = \frac{3}{7}$	$\overline{00}101._2$	
$\frac{22}{7}$	$2 + 2^3 + 2^4 (1 + 2^3 + 2^6 + \ldots) + 2^5 (1 + 2^3 + 2^6 + \ldots) = 2 + 2^3 + \frac{2^4}{1 - 2^3} + \frac{2^5}{1 - 2^3} = \frac{22}{7}$	$\overline{011}1010._2$	

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 $v_0^{\mathcal{B}_k}$ 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 $v_0^{\mathcal{B}_k}$ and $v_\perp$. Given $\mathcal{B}^* := (E^*, V)$, where $E_1^* = E_1 \cup (v_0^{\mathcal{B}_1}, v_\perp)$, $E_2^* = E_2 \cup (v_0^{\mathcal{B}_2}, v_\perp)$, and $E_3^* = E_3 \cup (v_0^{\mathcal{B}_3}, v_\perp)$, we define our triple-tree $\mathcal{T}$ as a clique-sum (shown in Figure 5):

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

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 $FC\left(S\left(\frac{7}{12}\right)\right)$, $FC\left(S\left(\frac{9}{20}\right)\right)$, and $FC\left(S\left(\frac{47}{60}\right)\right)$ 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 $v_0^{\mathcal{B}_k}$, 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 $\max\left(\Delta_2^{\mu}\right)$ parallelization, as discussed subsequently.

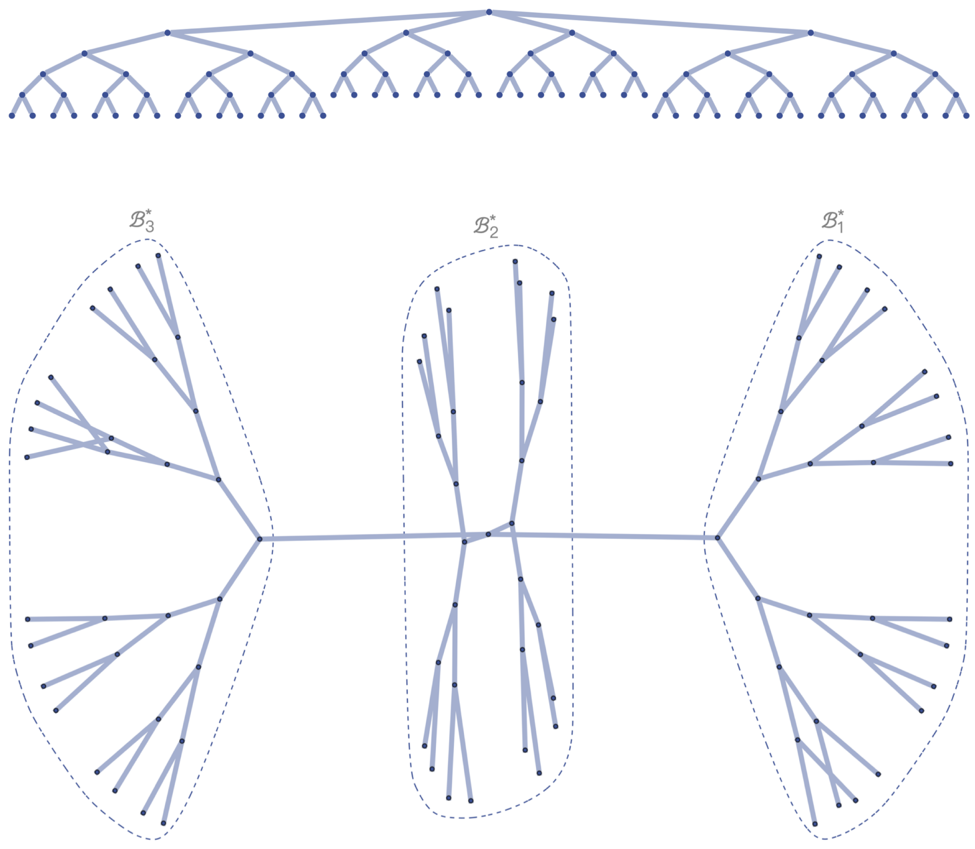


Figure 5: Top: an example illustration of $\mathcal{T}$. Bottom: the same illustration of $\mathcal{T}$ highlighting $\mathcal{B}_1^*$, $\mathcal{B}_2^*$, and $\mathcal{B}_3^*$. Note that these graph embeddings, for simplicity, collapse $v_\perp$ onto $v_0^{\mathcal{B}_2}$.

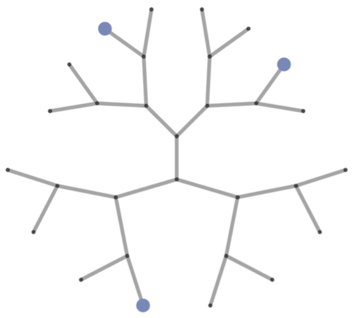
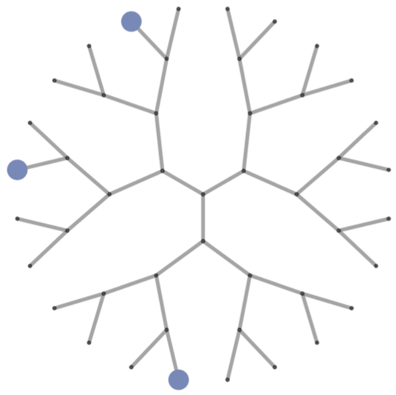
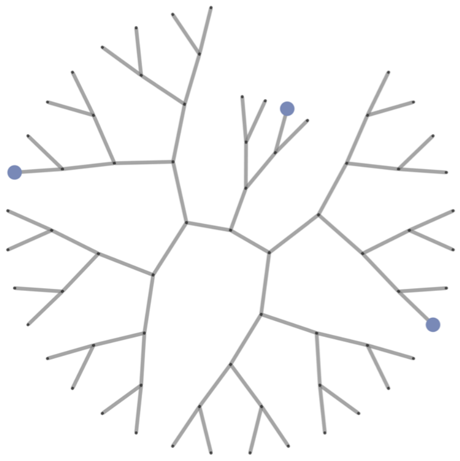
Rational	Triple–Tree Form	FC 1–2025 Encoding
$\frac{7}{12}$		▨⬜🟦▨⬜🟦🟦▨⬜🟦
$\frac{9}{20}$		▨⬜⬜🟦🟦▨⬜🟦▨⬜🟦
$\frac{47}{60}$		▨⬜🟦🟦🟦▨🟦⬜⬜⬜▨⬜🟦

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

4 The Processor

4.1 PR and 2AALU Clusters

The Hensel CPU architecture features a novel processor design, which is that of 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 Ξ_{PR} and a 2AALU cluster Ξ_{2AALU}. Operands are loaded to Ξ_{PR}, and arithmetic logic is performed in Ξ_{2AALU}. Both Ξ_{2AALU} and Ξ_{PR} belong to a common nested structure.

2AALU carriers are packaged within one another in nested fashion (all still being directly surface-mounted). 2AALUs feed their outputs to one another, with the carriers of the 2AALUs that they feed packaged within their carriers, recurrently, such that the carrier of a given 2AALU at level ℓ feeds its output to an 2AALU at level $\ell-1$ whose carrier is packaged within its own carrier, as visualized in Figure 8. The term "level" refers, technically, to the level in the processor circuit tree in which the 2AALU is situated, with a carrier for an 2AALU at level $\ell-1$ packaged within the carrier of the $\ell-1$ 2AALU that is its parent node in the circuit tree. (See the 2AALU report for further details.) The Ξ_{PR} is a collection of clustered processor registers, $\mathcal{C}_{\mathrm{PR}}$, whose containers are packaged in the lowest-level 2AALU carriers, as shown in Figure 7. Thus, the overall cluster consists of nested-packaged 2AALU carriers, with processor register carriers packaged within the innermost 2AALU carriers. 2AALUs positioned at different levels can modify different a_i, and, doing so simultaneously at different levels, parallelize Ξ_{2AALU} operations. This is achieved via execution of FC-2-2025-encoded instructions, as discussed subsequently.

Operands are loaded to Ξ_{PR} in distributed fashion. Given the $\mathcal{T}$-form of a given $\mathrm{FC}(\mathrm{S}(n))$, the individual $\mathcal{B}_k^*$ are loaded to distinct clustered processor registers in Ξ_{PR}. The PR cluster consists of a master PR, $\mathcal{M}_{\mathrm{PR}}$, as well as clustered PRs, written as $\mathcal{C}_{\mathrm{PR}}$. Together, they comprise the Ξ_{PR}, which we write in notation as a set of PRs $\Xi_{\mathrm{PR}} := \left(\bigcup_{i=1}^{\mathcal{N}} \mathcal{C}_{\mathrm{PR}_i}\right) \cup \mathcal{M}_{\mathrm{PR}}$ (where $\mathcal{N} < 2^{\mathrm{B}}$, with B being the architecture bit-width). The individual $\mathcal{B}_k^*$ are loaded to specific $\mathcal{C}_{\mathrm{PR}}$ according to individual encoding blocks called χ-IDs, discussed later in this report (and to greater effect in the Virtual Hensel report). A given χ-ID is loaded to the $\mathcal{C}_{\mathrm{PR}}$ whose address matches the χ-ID etnries. $\mathcal{M}_{\mathrm{PR}}$ is responsible for reassembling ("recompounding") these individual blocks back to a whole operand for storage purposes. Each $\mathcal{C}_{\mathrm{PR}}$ is loaded via receipt of an "activation input", which we'll term a π-sequence, issued by $\mathcal{M}_{\mathrm{PR}}$ as prompted by the LSU. A π-sequence both activates a given $\mathcal{C}_{\mathrm{PR}}$ and encodes information about the kind of χ-ID to be loaded (e.g., for a $\mathrm{R}_{\mathrm{FC}}, \mathrm{L}_{\mathrm{FC}}$, or T_{FC} block). Thus, to load an operand, the $\mathcal{M}_{\mathrm{PR}}$ loads the χ-IDs of its constituent blocks to the address-matching $\mathcal{C}_{\mathrm{PR}}$, and tells the $\mathcal{C}_{\mathrm{PR}}$ what kind of block is being loaded. Operands can be reassembled ("recompounded") with this information. The operands can also be subject to arithmetic operations, resulting in new χ-IDs, which are loaded to new $\mathcal{C}_{\mathrm{PR}}$ with matching addresses, and recompounded by the $\mathcal{M}_{\mathrm{PR}}$ to obtain the output.

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

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 2AALU carriers of differing sizes and mount them to the printed circuit board with carriers for 2AALUs at lower levels packaged within 2AALU carriers at higher levels, as shown in Figure 7. The process register carriers are, in turn, to be packaged inside the lowest-level 2AALU carriers, and thus the most deeply nested within the carrier packaging structure, as shown in Figure 8. It is this choice of inter-carrier packaging that gives the nested structure of the processor. (See the 2AALU report for further discussion.)

A Ξ_{2AALU} 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 Ξ_{PR} and level $\mathcal{L}$ storing the $\mathcal{M}_{\mathrm{PR}}/\mathcal{M}_{\mathrm{2AALU}}$), there are $2^{\mathcal{L}-\ell}$ 2AALUs $\{\mathcal{A}_{\ell,2^{\mathcal{L}-\ell}},\ldots,\mathcal{A}_{\ell,1}\}$ at each level ℓ. Each 2AALU carrier at level ℓ will contain 2 level-$(\ell-1)$ 2AALU carriers (with the exception of the $\mathcal{A}_{2,\mathrm{j}}$, whose carrier packages the the $\mathcal{C}_{\mathrm{PR}}$ carriers) and will be packaged alongside another 2AALU carrier by a 2AALU carrier at level $\ell+1$ (with the exception of $\{\mathcal{A}_{\mathcal{L}-1,1},\mathcal{A}_{\mathcal{L}-1,2}\}$, which are packaged within the $\mathcal{M}_{\mathrm{PR}}$ carrier). We can describe the nest structure of Ξ_{2AALU} in terms set membership, where, as a shorthand, $\mathcal{A}_{\ell,\mathrm{j}} \equiv \{\}_{\ell,\mathrm{j}}$. With this shorthand, we can write the nest structure as follows, beginning at $\ell = 2$ and moving outward by $\ell+1$:

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

4.3 Cluster Load-Store and Addressing

Operands are subject to load-store in distributed fashion, with the $\mathrm{R}_{\mathrm{FC(S(-))}}$, $\mathrm{L}_{\mathrm{FC(S(-))}}$, or $\mathrm{T}_{\mathrm{FC(S(-))}}$ blocks of an operand's encoding each loaded to a different $\mathcal{C}_{\mathrm{PR}}$. As described in greater detail in the Virtual Hensel report, IDs, called χ-IDs, are generated from these individual blocks, which are similar to the blocks themselves, but subject to a few coding tricks that permit a sizeable quantity of blocks to be subject to load-store by a relatively small number of processor registers. The address $\mathfrak{A}$ assigned to a given $\mathcal{C}_{\mathrm{PR}}$ is simply the χ-ID of the $\mathrm{R}_{\mathrm{FC(S(-))}}$, $\mathrm{L}_{\mathrm{FC(S(-))}}$, or $\mathrm{T}_{\mathrm{FC(S(-))}}$ encoding block load that it accepts. We denote a $\mathcal{C}_{\mathrm{PR}}$ with address $\mathfrak{A}$ as $\mathcal{C}_{\mathrm{PR}}(\mathfrak{A})$. Thus, distributed load-store is a matter of ID-address matching (i.e., χ-$\mathfrak{A}$ matching).

A given $\mathcal{C}_{\mathrm{PR}}$ is loaded by activating it with a π-sequence, which contains an encoding that distinguishes the χ-ID by its block type (i.e., $\mathrm{R}_{\mathrm{FC(S(-))}}$, $\mathrm{L}_{\mathrm{FC(S(-))}}$, or $\mathrm{T}_{\mathrm{FC(S(-))}}$), such that an operand, although being subject to load-store with its encoding broken down ("decompounded") into its constituent blocks, can nonetheless be recovered ("recompounded"). The LSU instructs the $\mathcal{M}_{\mathrm{PR}}$ to load a $\mathcal{C}_{\mathrm{PR}}$ as follows:

$$\lambda : (\chi,\pi) \to \mathcal{C}_{\mathrm{PR}}\left(\mathfrak{A}^{\chi}_{\pi}\right) \quad (16)$$

The λ mapping, as written above, is in fact a simplified description of what is done in practice. See the Virtual Hensel report for a finer description of loads, and the 2AALU report for a finer description of re-loads, which are applied to output operands.

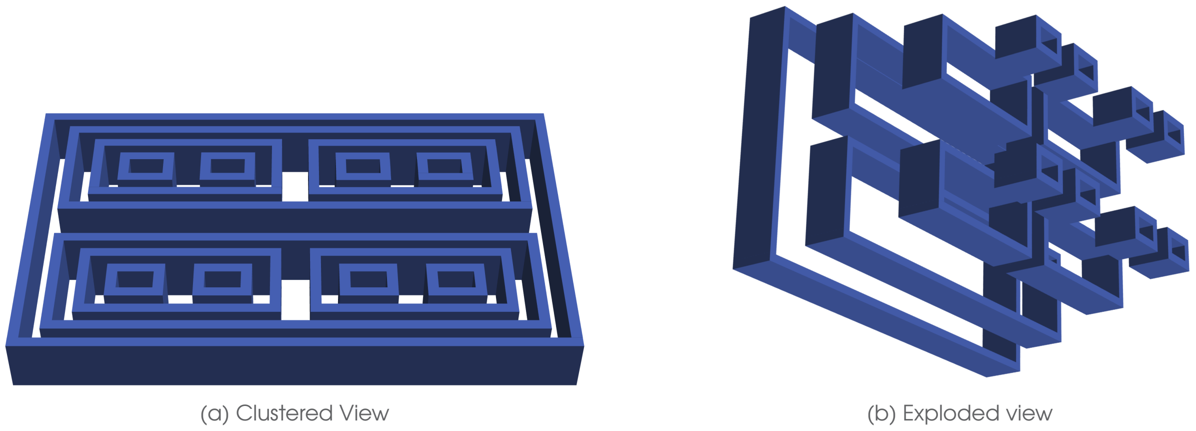


Figure 7: Illustration of the nested structure in Ξ_{2AALU}.

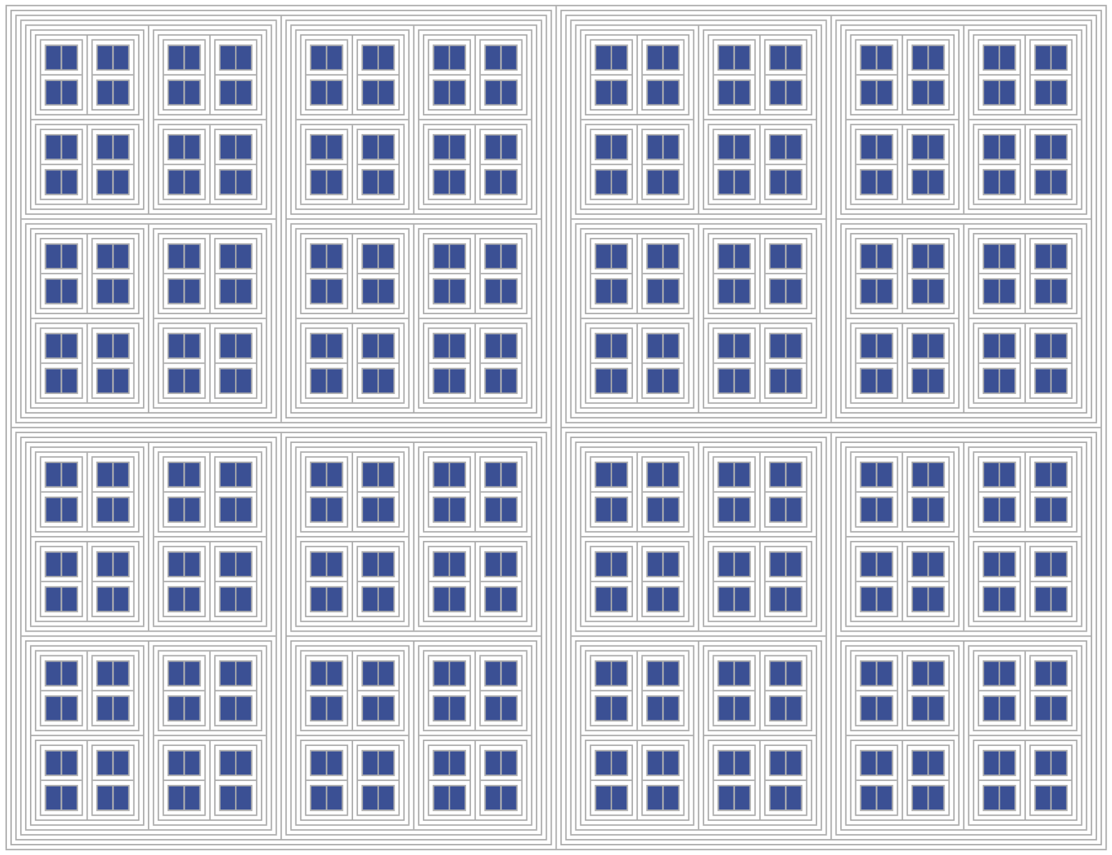
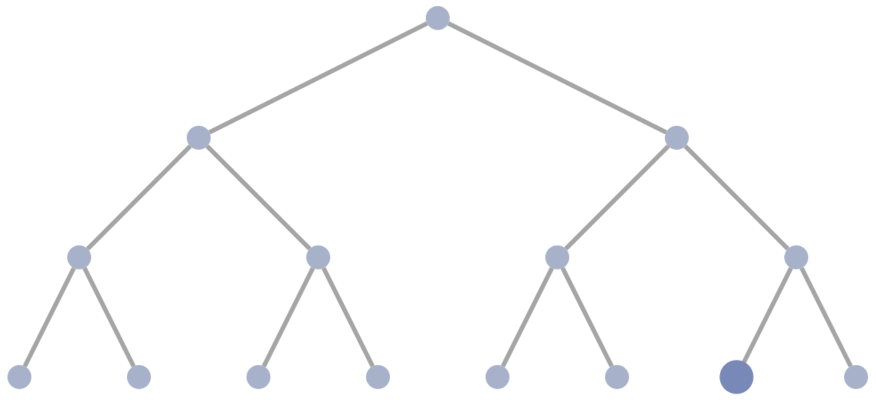


Figure 8: Cluster distribution of $\mathcal{C}_{PR}$ at the innermost level of Ξ_{2AALU}. (Note that the number of nested layers for Ξ_{2AALU} in this example is greater than in Figure 7, simply for purposes of visual variety.)



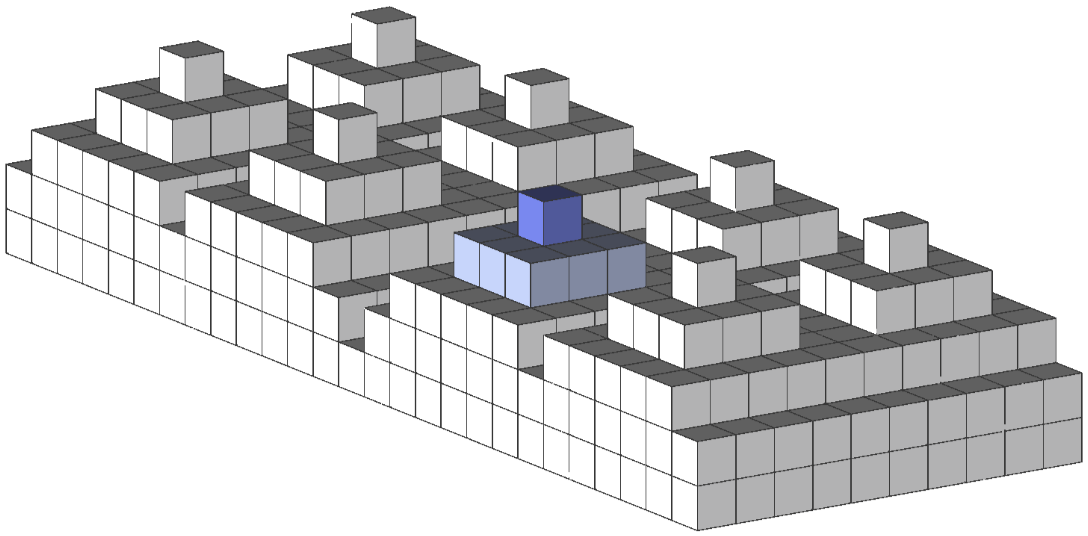


Figure 9: Illustration of the correspondence between $\mathcal{B}_k^*$-form tree structure and $\mathcal{C}_{\mathrm{PR}}$ cluster position. In this and other similar illustrations, the $\mathcal{C}_{\mathrm{PR}}$ are positioned at the "top", with layers "below" being 2AALU layers. Here, the top-most layers is nest-innermost in Ξ_{2AALU} (i.e., at level $\ell = 1$) and the bottom-most layer is nest-outermost in Ξ_{2AALU} (i.e., at level $\ell = \mathcal{L} - 1$).

5 Parallelized Arithmetic

Parallelization is to be performed by 2AALUs simultaneously in Ξ_{2AALU} 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 encoding 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 operands encoded in $\mathcal{T}$-form blocks, modification of individual blocks amounts to modification of χ-IDs, which, together (i.e., taking all blocks into account) amounts to wholesale modification of the operand.

Thus, given the distributed nature of load-store in the Hensel processor, a_i modification is performed block-wise as χ-modification. An operand is broken down into its constituent R_{FC}, L_{FC}, and T_{FC} blocks, encoded as χ-IDs, with arithmetic performed on the χ-IDs themselves. Modification of the many entries in χ-IDs is parallelized. For a binary arithmetic operation $\circ$ and operands $\mathsf{FC}(\mathsf{S}(\mathsf{q}))$ and $\mathsf{FC}(\mathsf{S}(\mathsf{p}))$, the computational depth of parallelization (i.e., the number of parallelized operations) will depend on $\mathsf{FC}(\mathsf{S}(\mathsf{q}\circ\mathsf{p}))$ and the number of modifications by which it differs from the inputs. When $\mathcal{M}_{\text{2AALU}}$ is alerted that the operands are $\mathsf{FC}(\mathsf{S}(\mathsf{q}))$ and $\mathsf{FC}(\mathsf{S}(\mathsf{p}))$ and the operation is $\circ$, it retrieves the instructions for computing $\mathsf{FC}(\mathsf{S}(\mathsf{q}\circ\mathsf{p}))$ from main memory. Such instructions are encoded according to the FC 2-2025 standard.

5.2 FC 2-2025 Encoding

Modification of entries in χ-IDs is nothing more than mere bit-flipping (e.g., $(1,1,1)\to(1,1,0)$). Indeed, as discussed in the 2AALU report, at the circuit level, modifications are executed in combinational logic by a given 2AALU by changing the input value (i.e., 0 or 1) for the logic gates at a particular level. At times, it is expedient to elide discussion of combinatorial logic and instead denote 2AALU operations in terms of "hop calculus", which, rather than explain the circuit-level combinational logic underpinning 2AALU operations, describes the effect of a 2AALU modifications, due to their effect on a given χ-ID, on corresponding changes in $\mathcal{C}_{\text{PR}}$ to which the χ-ID will be loaded. Indeed, a single 2AALU modification can alter a χ-ID such that its new address-matching $\mathcal{C}_{\text{PR}}$ is located in a notably different location in the processor cluster. However, because $\mathcal{C}_{\text{PR}}$ are located at the ends of the circuit in a specified manner (as discussed in the 2AALU report), the change-in-address consequent of 2AALU modifications is predictable, and very much related to the notion of 2-adic distance. A change-in-address can be thought of as a "hop" across the processor, and the 2AALU modification a "hop operation".

Each 2AALU in the Hensel circuit performs an operation and passes its output τ to the next 2AALU downstream in the circuit tree. A 2AALU can pass a τ via two kinds of hops. Because each 2AALU in the circuit tree (at level $\ell > 2$) has two children, i.e., $\{\mathcal{A}_{\ell,\text{I}},\mathcal{A}_{\ell,\text{II}}\}$ (and because, accordingly, each 2AALU carrier at level $\ell+1$ packages two 2AALU carriers at level ℓ), it is the case that τ can be passed according to one of two hops:

$$^{\tau}\mathsf{h}_{\ell}^{\sigma} : \mathcal{A}_{\ell,\text{I}} \to \mathcal{A}_{\ell,\text{II}} \tag{17}$$

or

$$^{\tau}\mathsf{h}_{\ell}^{\overline{\sigma}} : \mathcal{A}_{\ell,\text{II}} \to \mathcal{A}_{\ell,\text{I}} \tag{18}$$

Hereafter, we will write them simply as $\mathsf{h}_{\ell}^{\sigma}$ and $\mathsf{h}^{\overline{\sigma}_{\ell}}$, with τ implicit. (Alternatively, the 2AALU can perform no operation: a non-hop $\mathsf{h}_{\ell}^{\perp}$.)

The standard encoding for hop instructions, FC 2-2025, is as follows:

$$(\perp,\mathfrak{P}_{\text{R}},\perp,\mathfrak{P}_{\text{L}},\perp,\mathfrak{P}_{\text{T}}) \tag{19}$$

where $\mathfrak{P}_{\text{R}}$ is a sublist of instructions for R_{FC}; $\mathfrak{P}_{\text{L}}$, for L_{FC}; and $\mathfrak{P}_{\text{T}}$, for T_{FC}. The elements populating these $\mathfrak{P}_{(-)}$ sublists are $\mathsf{h}_{\ell}^{\sigma}$ and $\mathsf{h}_{\ell}^{\overline{\sigma}}$ (as well as $h_\ell^\perp$) where h_ℓ^σ is encoded as $(0,1)$ and $h_\ell^{\overline{\sigma}}$ 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. In practice, the Hensel performs arithmetic on χ-IDs, which are encoded according to the FC-3-2025 standard, as introduced in the Virtual Hensel report. In this report, for introductory purposes, we'll consider arithmetic on FC-1-2025 encoded operands. (We'll write "FC* 2-2025" to describe instructions on FC-1-2025 encoded operands, since, in practice, they are encoded for χ-IDs.) 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:

$$\begin{aligned}(\perp, \perp, (1,0), \perp, (0,1), \perp, (0,1), \perp, (0,0), \perp, \perp,\\ (0,1), \perp, (1,0), \perp, (1,0), \perp, (0,1),\\ , \perp, \perp)\end{aligned} \tag{20}$$

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

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

$$\begin{aligned}\mathfrak{P}\left(\mu(\mathsf{FC}(\mathsf{S}(q)))\right) &:= \mathcal{H}\\ &= \left(h_m^{(-)}, \ldots, h_2^{(-)}\right)\end{aligned} \tag{21}$$

where $\mathcal{L} - 1 \geq m$. Thus, a parallel computation performing a modification μ (written $\mathfrak{P}(\Xi(\mu(-)))$) of depth $\mathfrak{D} = \mathrm{Length}(\mathcal{H}) = m - 1$, will execute $(m-1)$-many $h_\ell^{(-)}$ operations. When χ-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 χ entries beginning at a_0 and operations beginning at $h_2^{(-)}$.

5.3 Parallelization and Non-Archimedean Distance

Hensel's nested structure is designed to optimally store, and efficiently compute with, numbers in $\mathbb{Q}_2$ in a manner commensurate with 2-adic arithmetic. For instance, the Hensel processor efficiently maximizes 2-adic output-operand distance over minimal operations because its nested structure is commensurate with 2-adic distance, which, unlike the archimedean case of $\mathbb{R}$, which is given with respect to a number line, is instead non-archimedean and gives a nested structure; the PR and 2AALU clusters are designed according to this structure.

Given the nested structure of Ξ_{2AALU}, a hop operation affects, at level $\ell = 1$, the board location of the output-ID-matching PR. Change-in-location is more distal when hops are performed at higher ℓ than at lower ℓ. Whereas at lower ℓ, hop operations can be performed all the while remaining nested within the same 2AALU packaging at higher ℓ, hops at higher ℓ in turn also affect all lower ℓ, due to nested packaging. Thus, one might suppose that the greater ℓ is, the greater the effect of a hop operation on the output. This would be rather inefficient, as it would imply that clearing greater arithmetic distances require π-sequence passing over greater board distances. However, with respect to 2-adic distance, one finds the opposite to be the case: the lower-ℓ hops have the greatest affect on the output. This is a consequence of triple-tree data-structure and nested-clustered design.

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

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

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:

$$\Delta_2^{\mu}(q) := |\mu(q) - q|_2 \tag{23}$$

Let's consider, in the parallelized case, the relationship between $\Delta_2^{\mathcal{H}}(-) = \sum_{\ell}^{\mathcal{L}-1} \Delta_2^{h_\ell^{(-)}}(-)$ and $\mathfrak{D}(\mathcal{H})$, the parallelization depth. Notably, greater $\mathfrak{D}(\mathcal{H})$, requiring $h_\ell^{(-)}$ at ever-greater ℓ, probes smaller 2-adic distances per $h_\ell^{(-)}$ operation, or to be more precise:

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

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

For instance, modification of a_0 has the greatest effect on Δ_2^{μ}. 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 $\Delta_2^{h_\ell^{(-)}} \geq 2^0$ (see Appendix).

At first glance, the above statement may seem surprising, for, in the usual case of a_i-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).

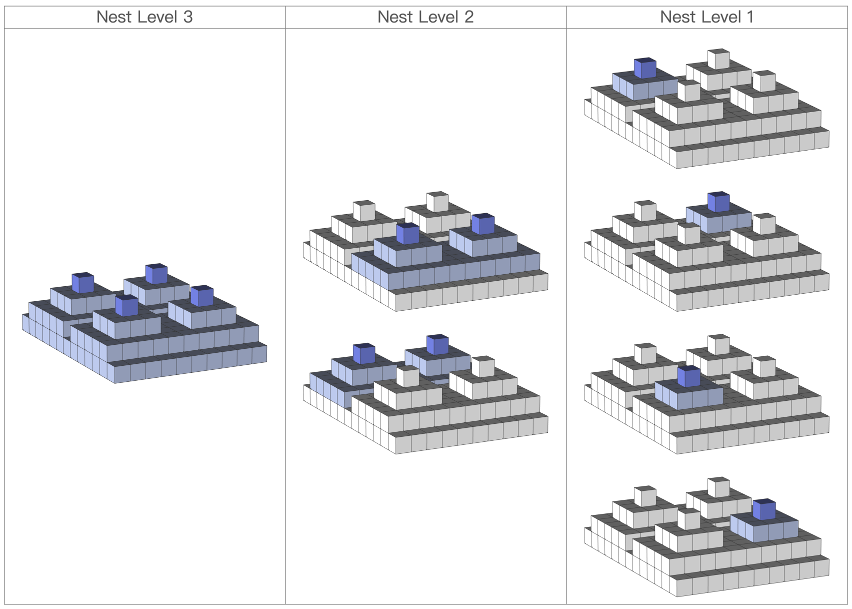


Figure 10: Table highlighting, per level, the 2AALUs a hop can reach (in light blue) and the PRs they can reach (in dark blue).

5.4 Max $\left(\Delta_2^{h_\ell}\right)$ Parallelization

If we look at $\Delta_2^{\mathcal{H}}$ per level ℓ, i.e., per $\Delta_2^{h_\ell^{(-)}}$, 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 $\Delta_2^{h_2^{(-)}}$ that can be cleared per level ℓ, as shown in Figure 11. As a consequence, given some $\mathcal{A}_{2,i}$ performing a $h_2^{(-)}$ hop, $\max\left(\Delta_2^{h_2^{(-)}}\right)$ is greater than $\max\left(\Delta_2^{h_{i>2}^{(-)}}\right)$. (See the Appendix for further discussion.)

Such is advantageous for parallelization, as it means that $\mathcal{A}_{\ell,j}$ are maximally level-efficient, per ℓ (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 $\Delta_2^{h_\ell^{(-)}}$-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.

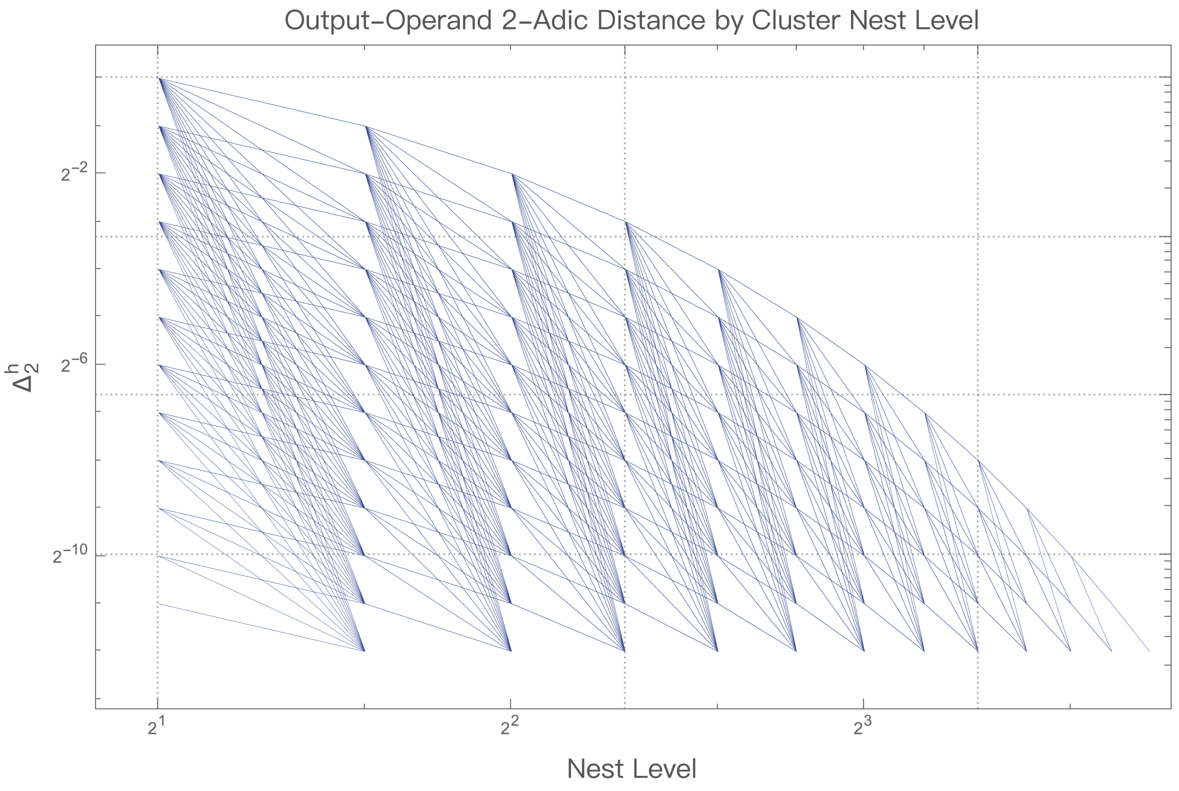


Figure 11: Plot showing possible $\Delta_2^{h_\ell^{(-)}}$ values per ℓ level, where $2 \leq \ell \leq 14$.

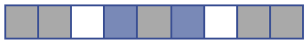
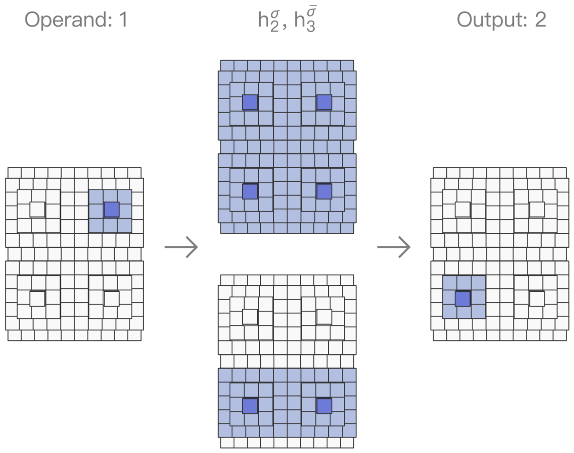
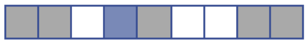
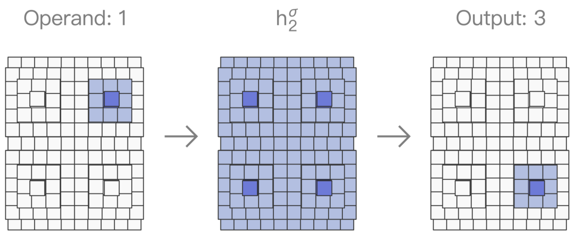
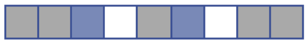
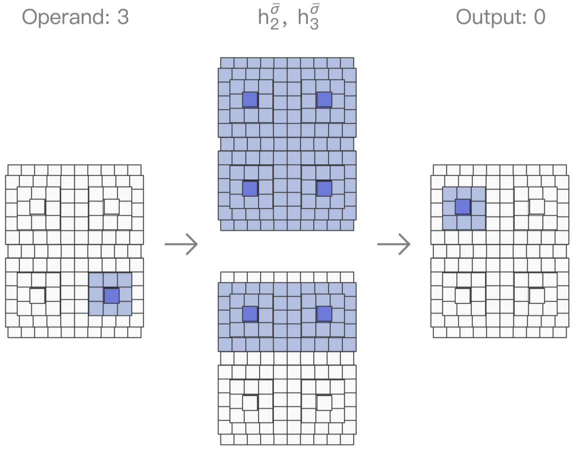
Sum	FC 2–2025 Instructions	2AALU Operations
1 + 1		 Operand: 1 $h_2^{\sigma}, h_3^{\bar{\sigma}}$ Output: 2
1 + 2		
3 – 3		 Operand: 3 $h_2^{\bar{\sigma}}, h_3^{\bar{\sigma}}$ Output: 0

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

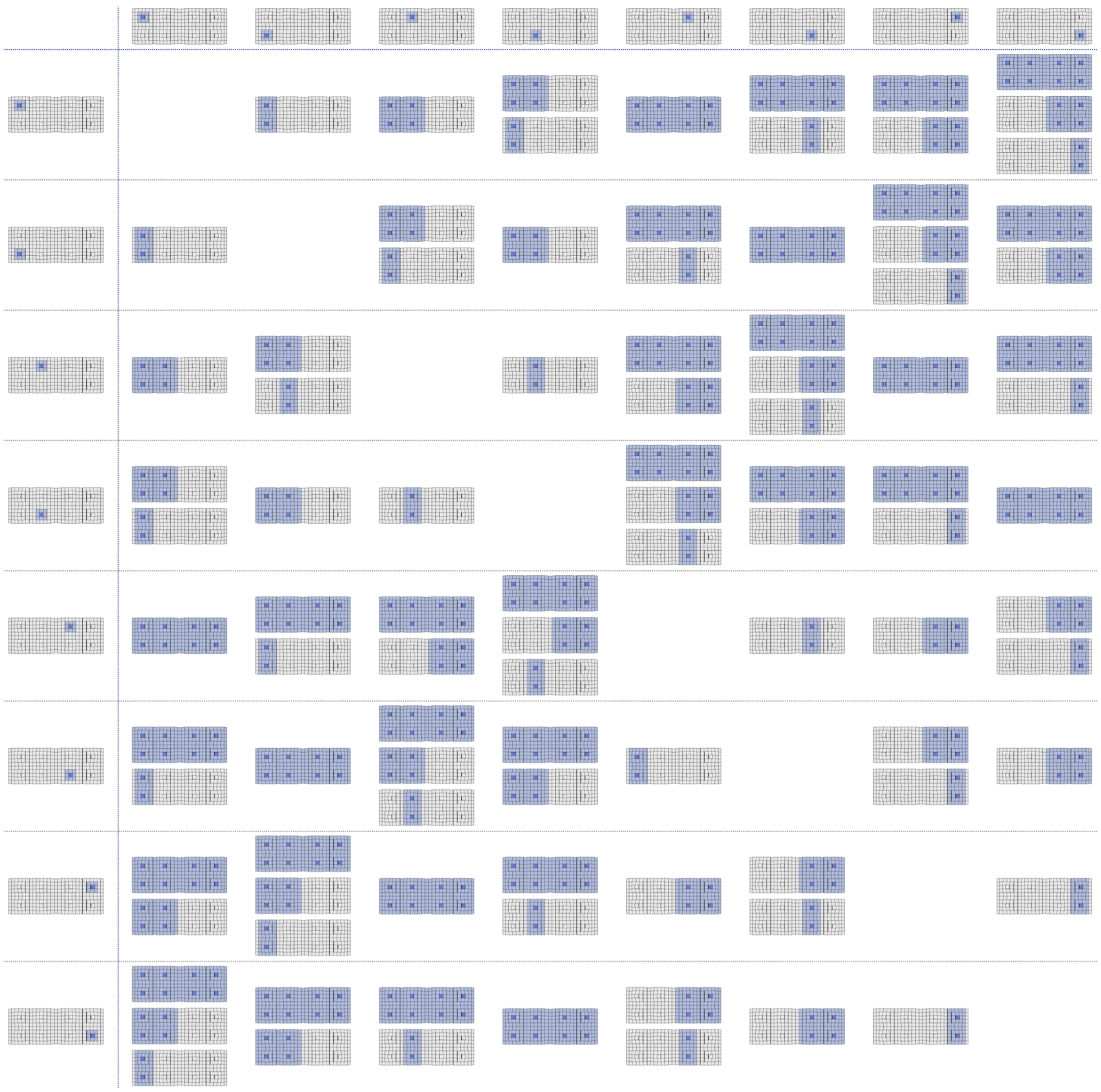


Figure 13: A $\mathfrak{P}$-form "instruction table", where the columns are $\mathcal{C}_{\mathrm{PR}}$ whose χ-addresses are the $\mathcal{T}$-form FC 1-2025 operand inputs $\{0, 1, 2, 3, 4, 5, 6, 7\} \in \mathbb{Q}_2$, and the rows are the same $\mathcal{C}_{\mathrm{PR}}$ whose χ-addresses are taken as outputs. Shown in the table are the $\mathsf{h}_\ell^{(-)}$ 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 $\max\left(\Delta_2^{\mu}\right)$ 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 Ξ_{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 $\mathfrak{P}\left(\Xi\left(FC\left(S\left(\mu\left(q\right)\right)\right)\right)\right)$ from the $\mathcal{T}$-form of $FC(S(q))$. The following 8×8 matrix gives the $\mathfrak{D}\left(\mathcal{H}\right)$ values for each:

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

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 Ξ_{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-$\mathfrak{D}\left(\mathcal{H}\right)$ 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 $\mathfrak{D}\left(\mathcal{H}\right)$ between hop-proximal and hop-distal $\mathcal{C}_{PR}$.

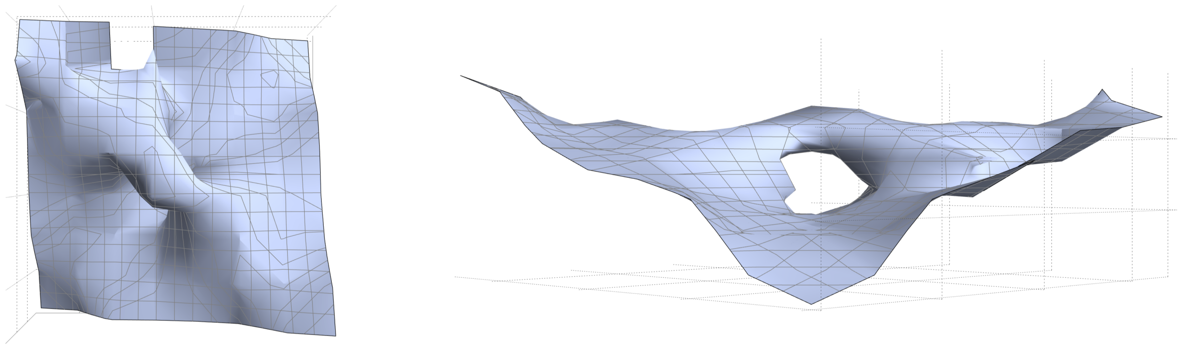


Figure 14: A 3D plot of $\mathfrak{X}$ seen from two viewing angles.

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 (χ, π)-Payloads

Error detection for operand encoding can be performed by taking advantage of the χ-ID system. Suppose a given encoding block is mis-transmitted (e.g., between a $\mathcal{C}_{\mathrm{PR}}$ and an $\mathcal{A}$, between the $\mathcal{A}$ and $\mathcal{M}_{\mathrm{PR}}$, etc.). Because the process begins by sending the appropriate χ from $\mathcal{M}_{\mathrm{PR}}$ to the appropriate $\mathcal{C}_{\mathrm{PR}}$, one could readily implement a check for operand-transmission consistency during Ξ_{2AALU} operations by requiring that the 2AALUs involved continue to transmit χ in their payloads by requiring that τ payloads become (χ, τ)-payloads, in which case FC-1-2025-form errors could be easily detected and located.

6.1.2 Δ_2^μ Trajectory-Defect Checks

Because 2AALUs at a given nest level ℓ can perform computations that affect Δ_2^μ by a certain 2-adic distance, an error in computation or transmission at any given level ℓ will be evident if it effectuates a change in $\mu(\mathfrak{q})$ beyond the Δ_2^μ range permissible for that level ℓ. Figure 15 gives an illustration.

6.1.3 Ξ_{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 Ξ_{2AALU} clusters whose outputs are sent to $\mathcal{M}_{\mathrm{PR}}$ upon clearing a vote. Note that such a process does not jeopardize parallelization, for the redundant Ξ_{2AALU} could compute in parallel with respect to one another. Nonetheless, an onus would be placed on the CU, main memory, and on the $\mathcal{M}_{\mathrm{PR}}$.

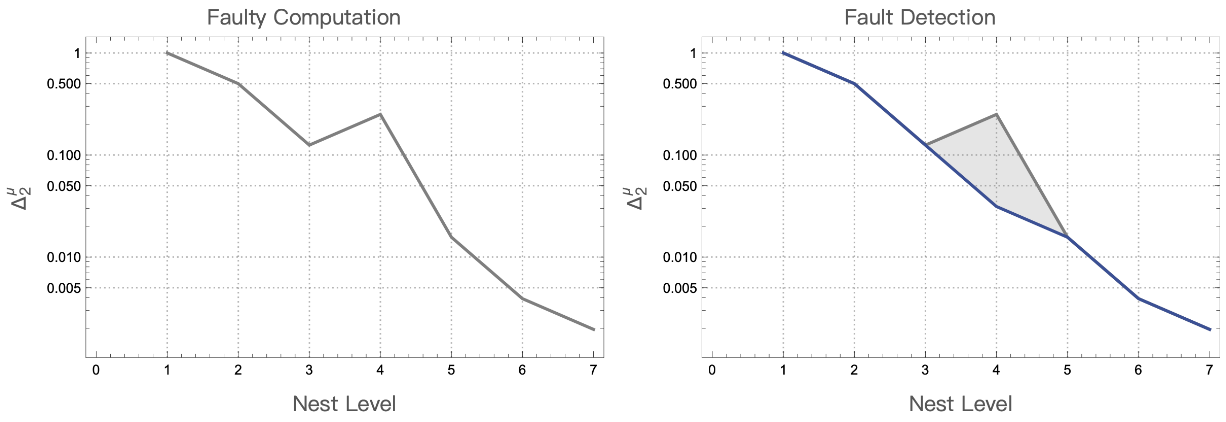


Figure 15: Illustration of Δ_2^μ trajectory-defect detection.

6.2 Supercomputing

The nested-clustered Ξ_{2AALU} 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

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

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

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

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

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

Ceteris paribus, a Hensel CPU is expected to require a Ξ_{2AALU} 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.

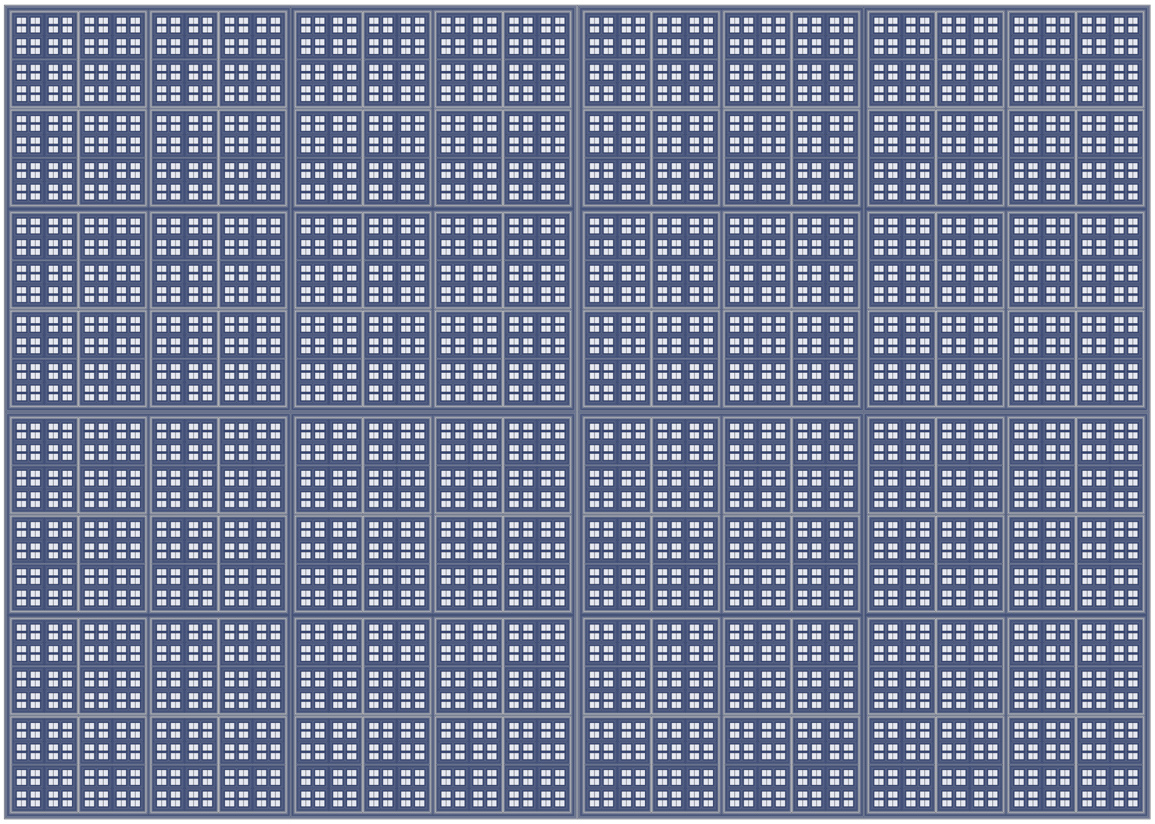


Figure 16: A 2AALU cluster of level depth $\mathcal{L} = 13$.

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 Ξ_{2AALU} 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.

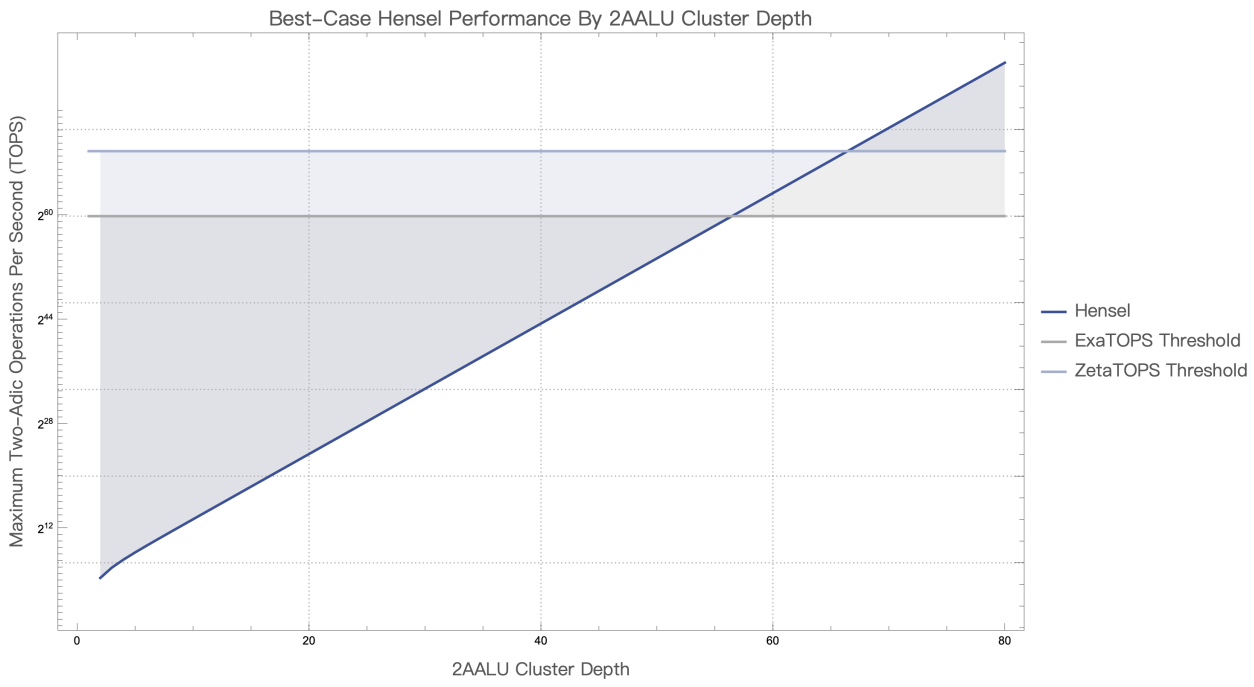


Figure 17: Best-case prediction of Ξ_{2AALU} level depth $\mathcal{L}$ (minus 2) required for exaTOPS and zetaTOPS performance.

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 $0 \leq \Delta_2^{h_\ell^{(-)}} \leq 1$.*

Proof. $\mathfrak{P}$-form instructions are given for Ξ_{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 Δ_2^{μ} too, by definition) thus bounded to $0 \leq |\quad|_2 \leq 1$, it is the case that a hop operation $h_\ell^{(-)}$ cannot clear an output-operand distance outside of the values $0 \leq \Delta_2^{h_\ell^{(-)}} \leq 1$. ☐

Theorem 2. *Comparing, at each ℓ, 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 Ξ_{2AALU} computation performing a modification μ on the operand, where the instruction list $\mathcal{H}$ contains operations $\left(h_m^{(-)}, \ldots, h_2^{(-)}\right)$ to be executed in parallel by 2AALUs in Ξ_{2AALU}, it is the case that*

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

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

Proof. In Ξ_{2AALU}, ℓ-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 $\max\left(\Delta_2^{h_\ell^{(-)}}(FC(S(q)))\right) = 2^{-\ell+2}$. (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 $h_{\ell+k}^{(-)}$ 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, $\max\left(\Delta_2^{h_{\ell+k}^{(-)}}(FC(S(q)))\right) = 2^{-\ell-k+2}$. It's trivial to see that $2^{-\ell-k+2} < 2^{-\ell+2}$ for all k where $\ell < k \leq m - \ell$. ☐

Theorem 3. *Given a list $\mathcal{H} = \left(h_m^{(-)}, \ldots, h_2^{(-)}\right)$ of $\mathfrak{P}$-form instructions for the modification μ 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 $\Delta_2^{\mathcal{H}}(FC(S(q))) = \sum_\ell^{\mathcal{L}-1} \Delta_2^{h_\ell^{(-)}}(FC(S(q)))$ be the total output-operand distance cleared by hop operations at each Ξ_{2AALU} level ℓ. At each level ℓ in Ξ_{2AALU}, a $h_\ell^{(-)}$ operation can decrease Δ_2^{Ξ} to $\Delta_2^{\Xi} - \Delta_2^{h_\ell^{(-)}}$. Due to Theorem 1, no $h_{\ell+k}^{(-)}$ operation can decrease Δ_2^{Ξ}

more than a $h_\ell^{(-)}$ operation. Thus, given $0 = \Delta_2^{\Xi} - \sum_{i=0}^{n} 2^{-i}$, at a given level ℓ, a $h_\ell^{(-)}$ operation can reduce Δ_2^{Ξ} by as much as $\left(\sum_{i=0}^{n} 2^{-i}\right) - 2^{-\ell+2} = \max\left(\Delta_2^{\Xi} - \Delta_2^{h_\ell^{(-)}}\right)$, and, with operations in $\mathcal{H}$ instructions being encoded by $h_\ell^{(-)}$, there is no smaller level-respecting list that reduces $\sum_{i=0}^{n} 2^{-i}$ to zero using $\Delta_2^{h_\ell^{(-)}}$ operations than

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

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

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

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

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

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

FC 1-2025 encodes T_{FC} and L_{FC}, which, being non-repeating, are thus unique due to the uniqueness of a_i. It remains to show that the the non-repeating a_i, encoded only once (i.e., without repetition) as R_{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 $R_{FC(S(E))} = (a_{w+L-1}, \dots, a_w)$. One can ask if there could exist some $R'_{FC(S(E))}$ such that, despite it beginning with a_w and not containing any repeating subsequences, $R'_{FC(S(E))} \neq R_{FC(S(E))}$ (thereby showing $R_{FC(S(E))}$ to be non-unique). This would require an instance in which $a_{w+i} - a'_{w+i+L} \neq 0$ (where $0 \le i \le L-1$), which would contradict the definition of $R'_{FC(S(E))}$ as repetitive. Thus, with a_i being unique, R_{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_{RC} and L_{RC}, which are unique for each $n \in \mathbb{Q}_2$ due to the uniqueness of a_i. □

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. □

SciSci Research

サイサイ・リサーチ

Published by SciSci Press