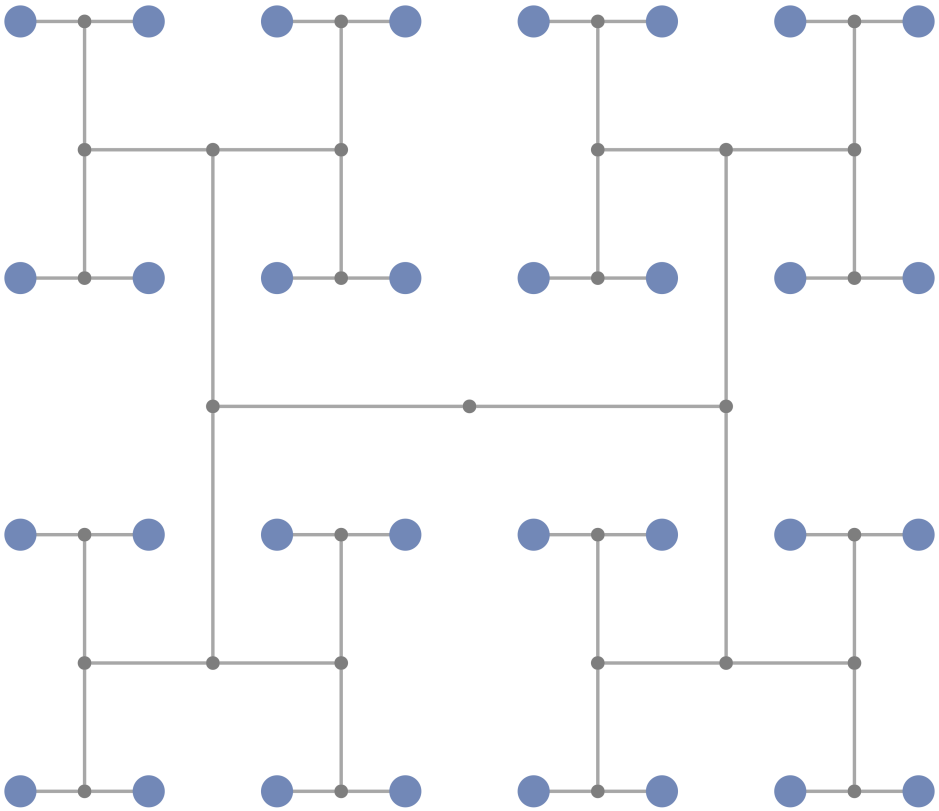
Inventions No. 3
sci-sci.org/load-store
DOI: 10.5281/zenodo.17231246

SciSci Research
サイサイ・リサーチ

Future Computing
コンフュー



Hensel CPU Load–Store Architecture

Distributed Register Design for Exact 2-adic 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 Load–Store Architecture: Distributed Register Design for Exact 2–adic Arithmetic. SciSci Inventions, 1(3). DOI: 10.5281/zenodo.17231246

Contents

1 Load-Store for 2-Adic Operands

The Hensel CPU processor stores operands in a distributed fashion in a cluster of processor registers, with the distribution, re-distribution, decompoundment, and compoundment of operands performed by the load-store unit (LSU). As discussed previously, Hensel performs arithmetic in $\mathbb{Q}_2$, with the distributed design of the register cluster intended to enhance computational efficiency by handlign operands in a manner commensurate with unique properties of p-adic fields, such as nonarchimedean distance. In what follows, a detailed description of how the LSU performs load-store in the distributed

Previous reports on the Hensel CPU and Virtual Hensel I, written to introduce and demonstrate architecture and functionality (particularly regarding the processor), have done so at the expense of giving a microarchitectural description of load-store. Instead, for purposes of convenience, a description of loads and reloads in the register cluster is given in terms of "hop calculus", a whose explanatory purpose is strictly high-level. Thus, it is necessary that a resource be provided on the fine details of the manner in which operands are handled during exact arithmetic.

2 The Hensel CPU Load-Store Architecture

2.1 Circuit Design

The Hensel processor performs loads and reloads in a largely parallelized fashion (plus a small $O(n)$ term). Processor load-store is performed within a LSU circuit. With operands loaded to processor registers distributed throughout the register cluster, the LSU also assumes a distributed form, consisting of "loader" operators, which amount to small gadgets in the LSU circuit that direct operand data to the appropriate register addresses.

The multi-level, nested design of the Hensel processor has been subject to extensive discussion in previous reports. Often, in discussion of the register cluster, one specifies its nest depth, $\mathcal{L}$. For instance, the Virtual Hensel I processor cluster is of nest depth $5\ (+2)$, i.e., it has five LSU levels and two additional levels (one for the clustered processor registers, $\mathcal{C}_{\mathsf{PR}}$, at level $\ell = 1$ and one for the master processor register and master LSU, $\mathcal{M}_{\mathsf{PR}}$ and $\mathcal{M}_{\mathsf{AALU}}$, at level $\ell = \mathcal{L}$). At the circuit level, a processor with nest depth $\mathcal{L}$ will have a circuit tree with $2 + 2^{\mathcal{L}-2}$ vertices (with each tree being a DAG, $\mathcal{M}_{\mathsf{PR}}$ being the source vertex, and the $\mathcal{C}_{\mathsf{PR}}$ being the sink vertices.)

LSU loaders perform load-store on operands according to FC-2-2025 instructions, which guide the loaders in modifying the entries in FC-3-2025 operand encodings. Loaders perform modifications in a multi-level fashion, with a processor of nest depth $\mathcal{L} = \mathtt{n}\ (+2)$ performing modifications on length-n operands, with each entry modified by a loader at a different level. Thus, each entry is modified by a different vertex at a different level in the circuit tree. We write the levels in descending order, beginning with the source, $\mathcal{M}_{\mathsf{PR}}$, at $\ell = \mathcal{L}$ and ending with the $\mathcal{C}_{\mathsf{PR}}$ sinks at $\ell = 1$. In terms of level-wise modification, the leftmost entry in the operand is modified by a loader at level $\ell = \mathcal{L} - 1$ and the rightmost entry is modified by a loader at level $\ell = 2$.

Loaders receive two kinds of inputs. Each circuit tree includes an initiator input Ψ, issued from level $\ell = \mathcal{L}$, which is always of value 1. Each loader also has its own modificatory input Φ, whose value depends on the FC-2-2025 instruction for the arithmetic at hand.

2.2 LSU Combinational Logic

Descending the LSU tree, one finds the first loader at level $\ell = \mathcal{L} - 1$. Ψ sends a 1 to both the XOR and XNOR gates of this $\ell = \mathcal{L} - 1$ loader. The loader gates also receive a Φ in-

put. One of the gates, fed this input pair from Ψ and Φ, will give a 1 as output, the other 0. Each gate provides input for a loader at the next level, giving a tree structure as illustrated in Figure 1. A loader is thus a tuple $(\text{XOR}, \text{XNOR}, \Phi)$, and the circuit is a tree built from loaders. The load computation continues downstream of the gate that gives a 1 output, terminating at a $\mathcal{C}_{\text{PR}}$ at level $\ell = 1$. For each loader, the load computation involves no more than feeding the received input from the loaders at the previous tree level, as well as the Φ input, to its two logic gates.

Let's consider an example where all the Φ inputs are 0. Given a circuit of tree depth $\mathcal{L}$, the output will be of length $\mathcal{L} - 2$ with entries all being of value 1. Let's see why. Suppose the nest depth is $\mathcal{L} = 5$. Here, the output is $(1, 1, 1)$. The leftmost entry is determined at tree level $\ell = 4$, the second at tree level $\ell = 3$, and the third entry determined at tree level $\ell = 2$. Ψ sends a 1, as always. Φ_4 will, in this case, send a 0. So, the inputs at $\ell = 4$ are 1 and 0. The XNOR gate will produce a 0, and the XOR gate will produce a 1, since $1 \odot 0 = 0$ and $1 \oplus 0 = 1$. Thus, the computation proceeds to the vertex downstream of the XOR gate. At the next level, $\ell = 3$, an XNOR gate and XOR gate are fed the 1 from the XOR gate at $\ell = 4$. We'll write these new gates as Γ_3^{XNOR} and Γ_3^{XOR}. So, Γ_3^{XNOR} and Γ_3^{XOR} receive a 1 from Γ_4^{XOR} and another input from Φ_3. In this case, Φ_3 will also give a 0 (since we're considering the example of all Φ values being 0). Thus, again, Γ_3^{XOR} will yield a 1, and Γ_3^{XNOR} will yield a 0. At $\ell = 2$, Γ_2^{XNOR} and Γ_2^{XOR} receive a 1 from Γ_3^{XOR} and a 0 from Φ_2, so Γ_2^{XNOR} will give a 0, and Γ_2^{XOR} will give a 1. Downstream of Γ_2^{XOR} is a $\mathcal{C}_{\text{PR}}$ whose address $\mathfrak{A}(\mathcal{C}_{\text{PR}})$ matches the output values in the tree: its address is $(1, 1, 1)$. Thus, with respect to the tree, we took a path of three consecutive XOR gates and terminated at the $(1, 1, 1)$-addressed processor register.

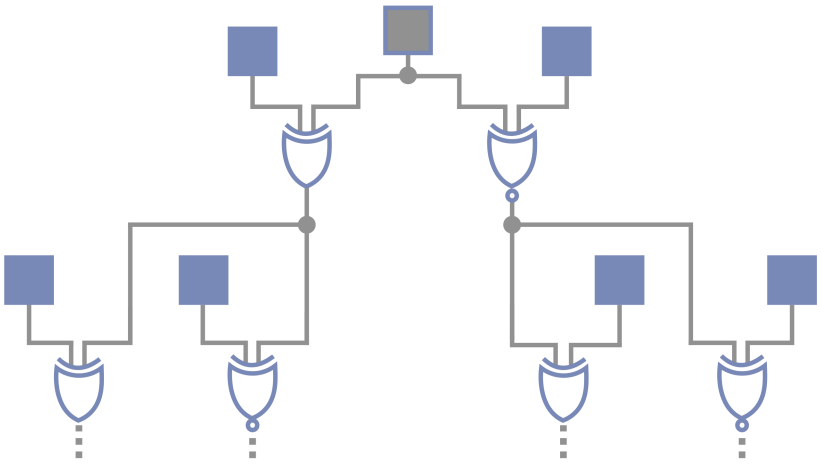


Figure 1: An illustration of an LSU circuit with three loaders. Each blue square is a Φ input, and the gray square is the Ψ input.

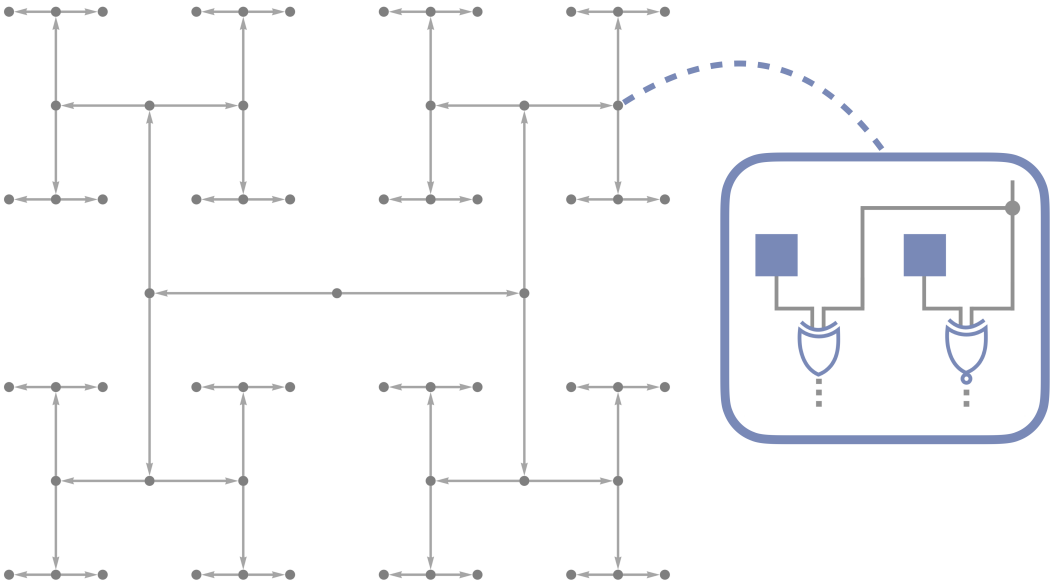


Figure 2: An illustration of the tree structure of the Hensel processor circuit, with each vertex its own loader, as visualized via a callout.

Next, suppose a Hensel arithmetic operation modifies $(1,1,1)$ to $(1,0,1)$. The LSU loads the output to the register cluster by "reloading" $(1,1,1)$ to $(1,0,1)$. The reload modification is effectuated by Φ_3: it gives a 1. Thus, Γ_3^{XOR} will give an output of 0, and Γ_3^{XNOR} will give an output of 1; now we've taken a new path down the tree, downstream of Γ_3^{XNOR}.

The $\mathcal{C}_{\text{PR}}$ are positioned at the ends of circuit tree paths such that, for a given path, its $\mathcal{C}_{\text{PR}}$ has an address whose entries are the same as the XOR gate values computed by the loaders that compose the path. In this case, the address is $(1,0,1)$, as were the XOR gate values. We can be more precise. Let $\mathfrak{C} = (V, A)$ be a directed graph with vertices V and directed edges given by ordered pairs A, where $\deg^+(v_i \in V) = 2$ and $\deg^-(v_i \in V) = 1$ (for $i \geq 1$), $\deg^-(v_i \in V) = 0$ for the case of $i = 0$ (i.e., the source vertex), and $\deg^+(v_k \in V) = 0$ where the v_k, given $|V| = 2 + \sum_{n=0}^{N=2} 2^n$, are the final 2^{N-2} vertices (i.e., the sinks). A path in the circuit tree can be written as $\Pi = (V', A')$, where $V' \subset V$, $A' \subset A$, and $A' = (v'_1, v'_2), (v'_2, v'_3), \ldots, (v'_{\mathcal{L}-1}, v'_{\mathcal{L}})$.

Each v'_i, for $2 \leq i \leq \mathcal{L} - 1$, corresponds to a loader. Let ${}^{\text{val}}\Gamma_\ell^{\text{XOR}}(v'_i)$ be the XOR-gate value of a given loader. Then, in A′, the vertex $v'_{\mathcal{L}}$ is a $\mathcal{C}_{\text{PR}}$ whose address $\mathfrak{A}$ is $\left({}^{\text{val}}\Gamma_\ell^{\text{XOR}}(v'_1), \ldots, {}^{\text{val}}\Gamma_\ell^{\text{XOR}}(v'_{\mathcal{L}-1})\right)$. This is by design. With each path Π terminating in a different and unique $\mathcal{C}_{\text{PR}}$, we can write the path as $\Pi\left(\mathcal{A}^{({}^{\text{val}}\Gamma_\ell^{\text{XOR}}(v'_1), \ldots, {}^{\text{val}}\Gamma_\ell^{\text{XOR}}(v'_{\mathcal{L}-1}))}\right)$. One thus gets χ-$\mathfrak{A}$ matching for free; χ-modification via Φ-perturbation directs the circuit tree path to the $\mathcal{C}_{\text{PR}}$ with the matching address.

As another example, suppose we have a nest depth of $\mathcal{L} = 7$. Let our input operand be $(0,1,1,0)$, with our arithmetic operation yielding $(1,0,1,1)$. In this case, we begin with $\Pi\left(\mathfrak{A}^{(0,1,1,0)}\right)$ and obtain $\Pi\left(\mathfrak{A}^{(1,0,1,1)}\right)$. The changes are executed by Φ_5, Φ_4, and Φ_2. We will call the effectuated change-of-path under χ-modification a Φ-perturbation. That is to say, χ-modification is performed at each level, and the overall change effectuated in the circuit tree is a Φ-perturbation, with χ-modification giving a new operand and Φ-perturbation giving a new Π.

3 Design for the LSU and Register Cluster

3.1 Nested Packaging

Naturally, a given path in the circuit tree will necessarily progress down multiple tree levels. Reloads amount to Φ_ℓ-perturbations applied at each level ℓ. The Hensel processor is itself of clustered form, designed to deliver Φ_ℓ-perturbations in parallel, by arranging loaders at multiple levels. The multilevel design is modular and nested. A given loader at level $\ell = n$ (where $2 \leq n \leq \mathcal{L} - 1$) is equipped with two logic gates, each of which, in turn, feeds into a loader at level $\ell = n - 1$. Hardware-wise, each $\ell = n$ loader is packaged in a carrier. In the case of the Hensel processor, these carriers are packaged inside one another, such that the carrier for a loader at level $\ell = n - 1$ fed by a loader at level $\ell = n$ is packaged within the carrier for the $\ell = n$ loader, with the $\ell = n - 1$ carrier necessarily of smaller size than the $\ell = n$ loader carrier. (It should be noted, however, that although being packaged within one another, all loader are nonetheless to be directly surface-mounted to the printed circuit board.) Nested packaging as such is illustrated in Figure 3.

The correspondence between carrier packaging and circuit tree structure is then as follows. A given loader in a circuit tree will be packaged in the carrier for the loader that is its parent vertex in the tree, as well as the other loader which shares the same parent vertex. For a given loader vertex $v \in V$ in $\mathfrak{C}$, let $\mathcal{K}$ be a set representing the carrier package for the loader. The nested packaging procedure for the Hensel CPU processor is then as follows:

$$(v_i, v_j) \in A \implies \mathcal{K}(v_j) \subset \mathcal{K}(v_i) \qquad (1)$$

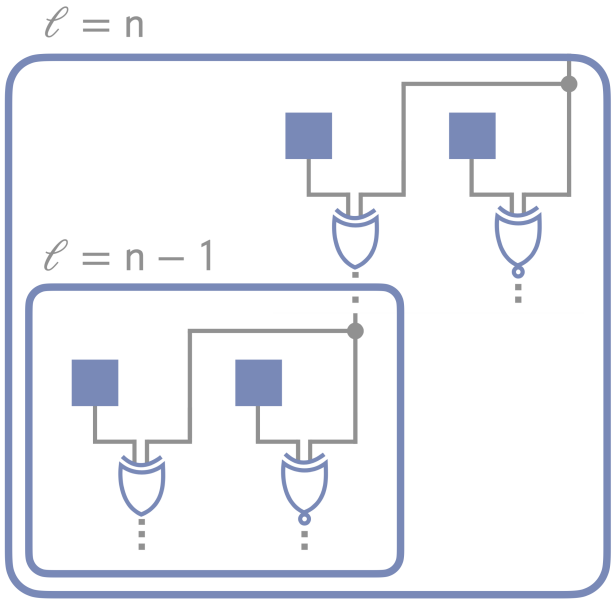


Figure 3: Illustration of nested packaging for loader carriers.

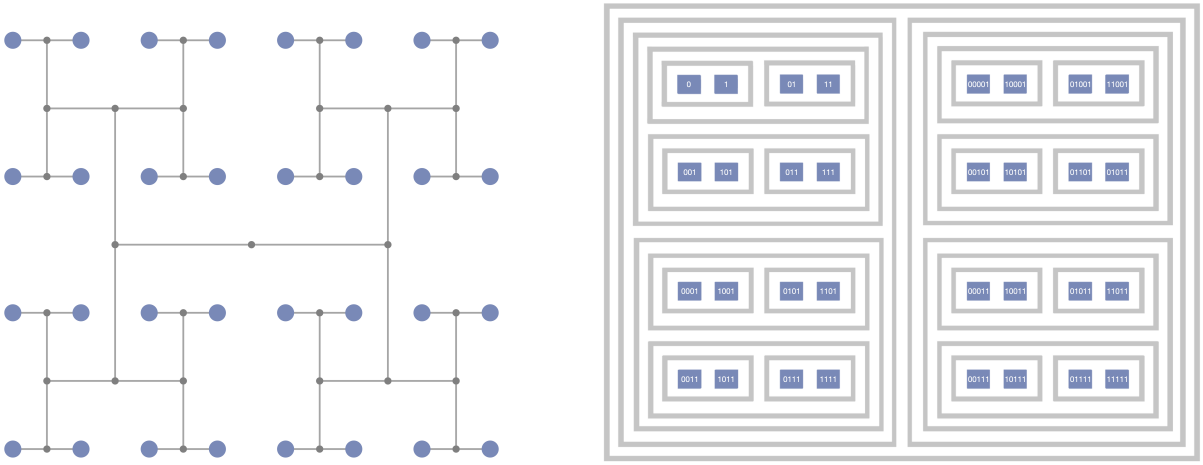


Figure 4: Comparison of loader combinational logic tree structure and processor register distribution. The processor registers are highlighted blue (as sink vertices in the tree on the left and as occupants of level $\ell = \mathcal{L}$ in the nested processor structure on the right).

As illustrated in Figure 4, there is a correspondence between the tree structure of loader combinational logic and the spatial distribution of processor registers in the Hensel processor cluster. This correspondence is the entirely by design: each path in the circuit tree terminates at a distinct processor register, and processor registers are positioned in the processor cluster according to their location in the circuit tree.

3.2 π-Sequence Passing

Following a computation, the new operand, the output, must in turn be taken up by the load-store architecture of the Hensel CPU, as described in previous reports. Not unlike the case of hop calculus, the term "π-sequence passing" has been used previously as a high-level shorthand for this process. Let us consider a more rarefied description herein. For expository gentleness, let us begin with the non-parallelized case where, at any given time, the master processor register, $\mathcal{M}_{\mathsf{PR}}$, is loaded with a particular operand. The FC-3-2025 encoding of this operand is decompounded into χ-IDs, which are then loaded to cluster processor registers with matching addresses. Each $\mathcal{C}_{\mathsf{PR}}$ has an addresses $\mathfrak{A}^{\chi}_{\pi}$, where χ is the matching ID and π is the π-sequence. $\mathcal{C}_{\mathsf{PR}}$-loading is described via load operations λ. (See the Virtual Hensel report).

Once an arithmetic computation in the processor has been completed, the new $\mathcal{C}_{\mathsf{PR}}$ at which the circuit tree path terminates under Φ-perturbation must in turn be loaded to the $\mathcal{M}_{\mathsf{PR}}$ in place of the original input operand. This is achieved via "re-loads" λ_{re}:

$$\lambda^{\mathsf{A}}_{\mathsf{re}} : \mathfrak{A}^{\chi}_{(*,-,-)} \to \mathfrak{A}^{\mu(\chi)}_{(*,-,-)} \tag{2}$$

$$\lambda^{\mathsf{B}}_{\mathsf{re}} : \mathfrak{A}^{\chi}_{(-,*,-)} \to \mathfrak{A}^{\mu(\chi)}_{(-,*,-)} \tag{3}$$

$$\lambda^{\mathsf{C}}_{\mathsf{re}} : \mathfrak{A}^{\chi}_{(-,-,*)} \to \mathfrak{A}^{\mu(\chi)}_{(-,-,*)} \tag{4}$$

$$\lambda^{\mathsf{D}}_{\mathsf{re}} : \mathfrak{A}^{\chi}_{(-,*,-)} \to \mathfrak{A}^{\mu(\chi)}_{(-,*,-)} \tag{5}$$

where μ is a modification, and $\mu(\chi)$ being the χ-modified ID encoding the output operand. (The significance of the subscripts, such as $(*,-,-)$ pertains to the R_{FC}, L_{FC}, and T_{FC} blocks in the FC-3-2025 encoding of the operand, i.e., its $\chi^{\mathfrak{d}}$-IDs, as described in the Virtual Hensel Report. A discussion of loads, i.e., λ mappings, can be found in the same report.)

Re-loads give a precise description of what in previous reports is described via the shorthand of "π-sequence passing": the π-sequences that were originally λ-loaded to $\mathcal{C}_{\mathrm{PR}}$ whose addresses matched the input $\chi^{\mathfrak{d}}$-IDs are now re-loaded (i.e., λ_{re}-loaded) to the output-$\chi^{\mathfrak{d}}$-matching $\mathcal{C}_{\mathrm{PR}}$, such that the output $\chi^{\mathfrak{d}}$-IDs, rather than the input $\chi^{\mathfrak{d}}$-IDs, are now loaded to $\mathcal{M}_{\mathrm{PR}}$. Following these re-loading operations, the output $\chi^{\mathfrak{d}}$-IDs in $\mathcal{M}_{\mathrm{PR}}$ are recompounded and sent to the load-store unit (as discussed in the Virtual Hensel report).

3.3 Hop Calculus

As discussed, the reload modifications are applied via change of Φ_ℓ inputs. The processor executes all Φ_ℓ in parallel. Then, the processor runs the path computation of the Φ-perturbation, which is not parallel, but imposes but a small additional linear term, $O(n)$, where $n = \mathcal{L} - 1$. These Φ_ℓ are the so-called "hop operations" in hop calculus; Φ_ℓ inputs cause the path to "hop" from the input-terminating path to the output-terminating path. One gets a new Π by changing at least one Φ_ℓ value; thus, one can describe loader modifications in terms of Φ inputs alone, i.e., in terms of hops, sparing the details of its underlying combinational logic. Thus, it is convenient to describe register reloading in terms of hop calculus. For instance, returning to the previous example of the computation yielding $\Pi\left(\mathfrak{A}^{(1,0,1,1)}\right)$ from $\Pi\left(\mathfrak{A}^{(0,1,1,0)}\right)$, the Φ_ℓ-perturbations can be described solely in terms of hops: $\left(h_5^{\sigma}, h_4^{\overline{\sigma}}, h_2^{\sigma}\right)$. Here, a h_ℓ^{σ} hop is a Φ_ℓ input giving a 1 and a $h_\ell^{\overline{\sigma}}$ hop is a Φ_ℓ input giving a 0. (See the Virtual Hensel report for a review of hop calculus notation.)

We can more precisely describe the relationships between hops and Φ_ℓ inputs. Suppose we have a $\chi^{\mathfrak{d}}$-ID loaded to a given cluster processor register. Let $\chi^{\mathfrak{d}}_{(-,-,*)} = (1,1,1,0,1)$, which is the T_{FC} block for the FC-3-2025 encoding of $\frac{29}{32}$. Suppose our arithmetic operation modifies this $\chi^{\mathfrak{d}}$ block to $(1,0,0,0,0)$. (For instance, if the arithmetic is $\frac{29}{32} - \frac{13}{32}$, the output encoding will include the single block $\chi^{\mathfrak{d}}_{(-,-,*)} = (1,0,0,0,0)$.) Looking at the IDs, one sees that three modifications are in order, namely at Φ_6, Φ_4, and Φ_3. Figure 5 illustrates both tree paths, with the Π segments that differ by Φ_ℓ input highlighted purple.

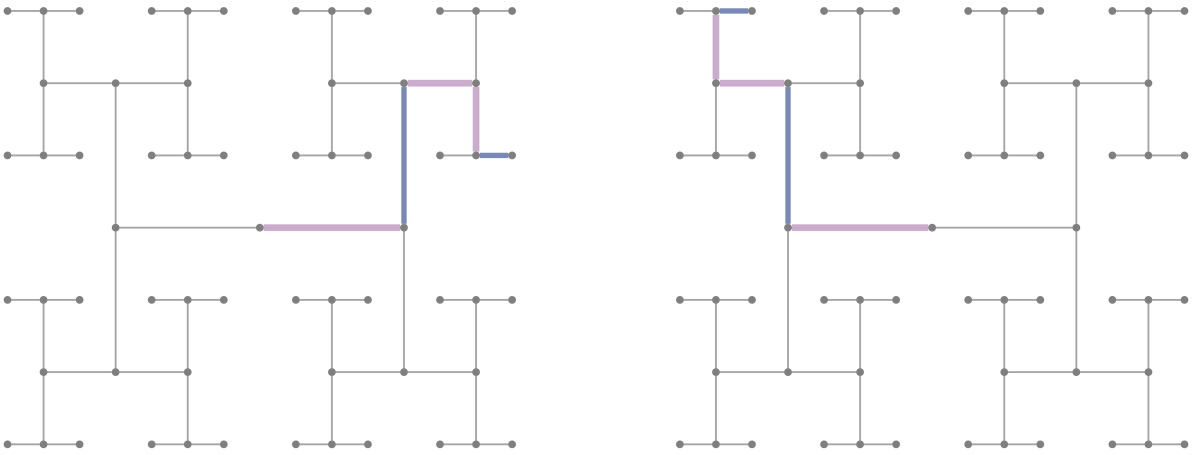


Figure 5: An illustration of $\Pi\left(\mathfrak{A}^{(1,1,1,0,1)}_{-,-,*}\right)$ and $\Pi\left(\mathfrak{A}^{(1,0,0,0,0)}_{-,-,*}\right)$. Φ_6, Φ_4, Φ_3 are highlighted purple.

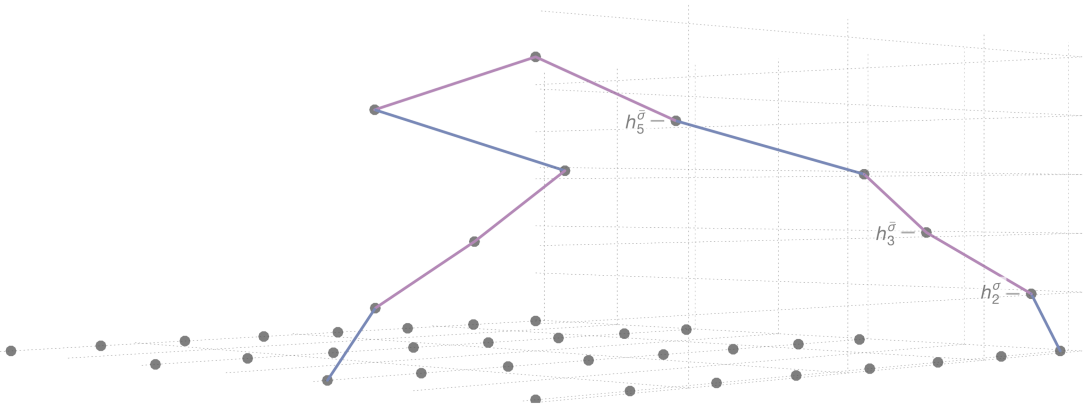


Figure 6: A three-dimensional illustration of Figure 5, with ℓ levels distinguished along the z axis.

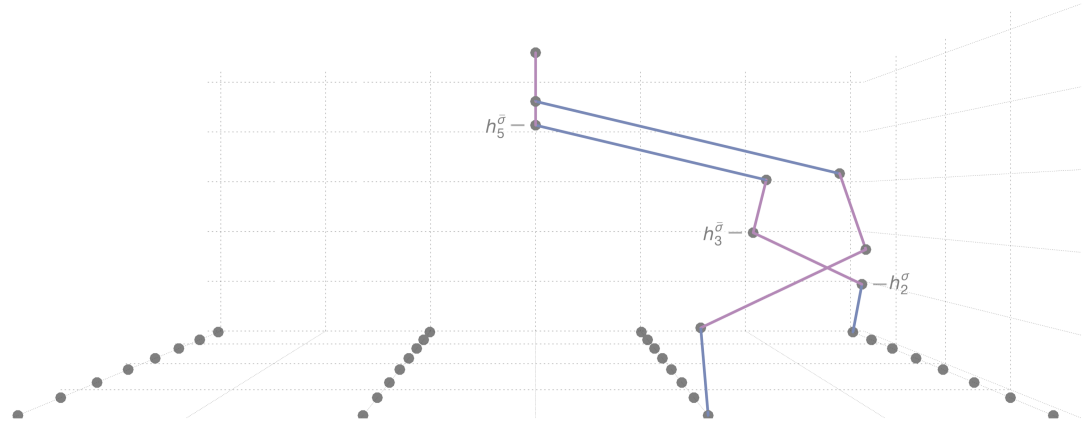


Figure 7: Figure 5 as seen from a different angle.

As one can see, Φ_6, Φ_4, and Φ_3 divert the tree path from the processor register with address $\mathfrak{A}^{(1,1,1,0,1)}$ to that with address $\mathfrak{A}^{(1,0,0,0,0)}$. Φ_6 directs the tree leftward, rather than rightward; Φ_4 also directs the tree leftward rather than rightward; and Φ_3 directs the tree upward rather than downward. Figures 6 and 7 give three-dimensional illustrations, with ℓ-level parameterized by the z-axis, and the points at level $\ell = 1$ being the $\mathcal{C}_{\text{PR}}$ at which the computations terminate. These figures illustrate the effect of Φ-perturbations on both circuit tree paths and the processor register destinations.

3.4 Parallelization and Scaling

Thus far, our descriptions have been restricted to a Hensel processor with a single circuit tree. However, there is no reason why multiple trees cannot be employed, so long as each $\mathcal{C}_{\text{PR}}$ is designed to accept multiple inputs and the loader carriers are designed to package multiple trees. Multiple trees would allow for parallelization of FC-2-2025 instruction execution; the processor could perform arithmetic on multiple inputs via multiple circuit trees.

SciSci Research

ｷｬﾄｷｬﾄ・ﾆｷｰﾙ

Published by SciSci Press