Energy Efficient Convolutions with Temporal Arithmetic

Abstract

Convolution is an important operation at the heart of many applications, including image processing, object detection, and neural networks. While data movement and coordination operations continue to be important areas for optimization in general-purpose architectures, for computation fused with sensor operation, the underlying multiply-accumulate (MAC) operations dominate power consumption. Non-traditional data encoding has been shown to reduce the energy consumption of this arithmetic, with options including everything from reduced-precision floating point to fully stochastic operation, but all of these approaches start with the assumption that a complete analog-to-digital conversion (ADC) has already been done for each pixel. While analog-to-time converters have been shown to use less energy, arithmetically manipulating temporally encoded signals beyond simple min, max, and delay operations has not previously been possible, meaning operations such as convolution have been out of reach. In this paper we show that arithmetic manipulation of temporally encoded signals is possible, practical to implement, and extremely energy efficient.

The core of this new approach is a negative log transformation of the traditional numeric space into a ‘delay space’ where scaling (multiplication) becomes delay (addition in time). The challenge lies in dealing with addition and subtraction. We show these operations can also be done directly in this negative log delay space, that the associative and commutative properties still apply to the transformed operations, and that accurate approximations can be built efficiently in hardware using delay elements and basic CMOS logic elements. Furthermore, we show that these operations can be chained together in space or operated recurrently in time. This approach fits naturally into the staged ADC readout inherent to most modern cameras. To evaluate our approach, we develop a software system that automatically transforms traditional convolutions into delay space architectures. The resulting system is used to analyze and balance error from both a new temporal equivalent of quantization and delay element noise, resulting in designs that improve the energy per pixel of each convolution frame by 8x compared to a state-of-the-art.

1. Introduction

Convolution is a critical operation for image processing, being the basis for feature extraction [29], filters [36], and edge detection [48]. The rise of visual sensor networks [10] and convolutional neural networks (CNNs) [22] have further amplified the importance of convolutions. In the domain of sensor-embedded computation specifically, where convolution operations are typically local and resident, energy consumption is dominated by the cost of converting the data into a form that can be manipulated digitally and the cost of multiply and accumulate (MAC) operations [40].

Technology scaling helps reduce energy consumption but the slowing of traditional computational scaling, coupled with ever increasing sensor array densities, creates opportunities for non-traditional computing paradigms. While this can be as simple as reduced-precision floating point [18], more radical approaches include stochastic computing [45, 46] and race logic [25], which experiment with alternative data encodings. These data encodings are of particular interest because they leverage the electrical behavior of basic binary logic gates in new ways. Race logic encodes information into the timing of a voltage edge and relies on four operations: minimum, maximum, (with AND and OR respectively), delay, and inhibit [25, 41] to perform computation. These operations are logically complete [39] and have a near minimal activity factor [26], creating the potential for incredibly energy efficient computation. However, to the best of our knowledge no prior work has demonstrated efficient arithmetic operations with race logic primitives, limiting the general applicability of race logic.

A primary contribution of this work is a new temporal value encoding — one that fully leverages the existing race-logic circuits, but that also allows for efficient arithmetic — while still representing values as a single edge.

The key insight is to apply a mathematical transformation under which addition and multiplication are co-transformed with the encoded values, forming a new mathematical ring over delays. Data values in traditional importance space are converted into a delay space with a rising edge occurring after a delay equal to the negative log of the value. Multiplication in importance space becomes simple addition in delay space, which means that multiplication can be implemented with a simple delay operation. Addition, however, becomes a negative log sum exponential (nLSE) function. While the nLSE function initially seems challenging to implement, we show that the function can be efficiently approximated with nothing more than min, max, and delay.

While delay space is interesting theoretically and opens many new avenues for temporal computation, it is also practically implementable, tolerant to noise, and highly energy efficient in practice. To demonstrate these points, we design and evaluate a near-sensor architecture operating completely in delay space, from sensor activation through convolution, as shown in Figure 1.
This proposed architecture matches the iterative row-by-row read-out of most sensor arrays with a temporal scaling and summation that can be applied iteratively as each new row is read. At the core of this convolution system is an automated transformation that converts the MAC operations into nLSE and delay units in a novel recurrence architecture. This work presents the following contributions:

- A new data encoding that transforms the linear operations of convolution into new operations in an inverted logarithmic delay space, significantly expanding the computational potential of temporal computing.
- Efficient approximations for the new “soft” operations required in delay space, using existing “hard” temporal logic primitives, and recurrent hardware implementations of those approximations to iteratively perform delay-coded summations.
- An architectural evaluation tool for our convolution architecture and the hardware implementations of these temporal computations. We demonstrate the utility of this approach by embedding these linear operations within the natural temporal staging of pixel read-out.
- We quantify the impact of various noise factors on approximation accuracy and show that realistic designs can operate with energy efficiencies 8x better than prior work in the area.

We start with description of the encoding, its properties, and hardware implementation (Sections 2 and 3) before moving on to a more complete architecture and evaluation (Sections 4 and 5), finishing with connections to other work and final conclusions (Sections 6 and 7).

---

2. Delay Space Arithmetic

In race logic, a signal’s time of arrival – when a rising or falling edge occurs – encodes the signal’s value, rather than the signal’s voltage level. Using just four basic operations on those signals, first arrival (FA), last arrival (LA), INHIBIT, and DELAY [25, 41], arbitrary temporal functions can be constructed [39]. If one thinks about the delay in time as linearly encoding a value (as has been assumed by prior work), then FA and LA execute min and max functions respectively, while the delay performs simple addition. This makes some intuitive sense as a signal computed as FA(τ1, DELAY(τ2, δ)) will appear at time min(τ1, τ2 + δ).

This encoding has the advantage of a very simple hardware mapping, with FA and LA on rising edges being implemented by simple OR and AND gates respectively. While not corresponding exactly to existing logic gates, INHIBIT only requires two transistors [41], and there are a myriad of hardware delay elements [28].

In addition to being a complete temporal logic, race logic has been shown to efficiently implement shortest path graph algorithms, decision trees, sorting networks, and other useful constructions [25, 41, 39]. However, efficient implementations of more general arithmetic operations with race logic primitives has never been demonstrated, in part due to the complexity of multiplication and the inability to represent negative numbers or perform subtraction. Ideally we would be able to achieve the following properties:

1. **Operations directly on encoded form**: Given a real number we need a clear way to get into and out of this encoding coupled with a way to perform the operations necessary for convolution (addition, subtraction, and multiplication) directly in the encoded form, without unnecessary conversions. More formally we could say we are looking for a bijective ring homomorphism of the reals.

2. **Important values early**: Traditionally, a signal of larger magnitude is represented with a larger number. When dealing with delays, it would be far more natural for highly important values to be encoded as shorter durations of time, so that less important contributions can be truncated at any time. The more “excited” the system, the smaller the delay and hence the larger the importance – meaning the events that occur first are those that carry the most weight.

3. **Broader dynamic range**: From image processing to machine learning, the ability to stretch beyond the constraints of linear encoding is very useful. This is particularly critical when dealing with delays because the execution time and energy consumption is coupled to the data representation. Most prior “unary” schemes map values linearly, which makes it difficult to deal with very large and very small values in the same computation. An ideal solution would allow a broad range of values to be operated on in a way similar in spirit to a floating point representations, without introducing the problems of normalization.
We propose that a new encoding meeting all of these requirements is a negative log mapping, where a value \( x \) in the original convolution maps to a signal that has a rising edge after a time delay of \( x' \) where

\[
x' = -\ln(x)
\]

Equation 1 and logarithmic identities. A delay space multiplication can be performed using simple addition. Delay space addition and subtraction map to the negative log sum exponential (nLSE) and negative log difference exponential (nLDE) functions respectively:

\[
\begin{align*}
    x \cdot y &\rightarrow x' + y' \\
    x + y &\rightarrow -\ln(e^{-x'} + e^{-y'}) = nLSE(x', y') \\
    x - y &\rightarrow -\ln(e^{-x'} - e^{-y'}) = nLDE(x', y')
\end{align*}
\]

2.1. Approximate Delay Space Addition

At first glance this negative log space seems awkward – the nLSE and nLDE functions in particular seem difficult to efficiently implement in hardware. But delay space has some nice properties which we can exploit. First, as values approach infinity in importance space, they approach zero time in delay space, so more important values have less delay.

Second, because it is a logarithmic encoding, the value space gives reasonable encodings to a far wider range of values. Furthermore we can re-scale our values by simply shifting the reference point for the delay, because addition or subtraction in delay space is the same as multiplicative scaling in importance space. This means we have a much improved range to operate in. Finally, and most critically from an implementation standpoint, the nLSE function is a form of “soft min” operation (where LSE is the “real soft max” used commonly in machine learning [3]) and is associative, commutative, and addition distributes through it as nLSE(\(a + \delta, b + \delta\)) = nLSE(\(a, b\)) + \(\delta\). These properties, along with its bounds, allow us to come up with arbitrarily tight approximations for nLSE using only min, max, and delay.

Figure 2 shows the nLSE function (Equation 4). The function is bounded from above by min(\(x', y'\)) and at the most extreme points, where one of \(x'\) or \(y'\) is much larger than the other, the behavior of nLSE(\(x', y'\)) converges to min(\(x', y'\)). However, as \(x'\) and \(y'\) become closer in value, nLSE deviates further and further from min, with worst-case error \(-\ln(2)\) when \(x' = y'\).
While addition and multiplication give us most of what we need to perform convolution, an additive inverse is required to handle the negative constants common to many convolution kernels and complete the mathematical ring. To enable subtraction and negative value representation we adopt an approach similar to memristive crossbars [6] or dual rail computing [42], where we split all numbers into two non-negative pairs, \((x_{pos}, x_{neg})\). If the value is positive, \(x_{pos}\) equals the value and \(x_{neg}\) is 0. For a negative value, \(x_{pos}\) is 0 and \(x_{neg}\) is the absolute value. For zero values, \(x_{pos}\) and \(x_{neg}\) are both zero.

With this representation, subtraction simply becomes addition with the \(x_{neg}\) field of the second operand, but we must re-normalize the result to ensure that at least one of \((x_{pos}, x_{neg})\) is zero. This re-normalization only has to occur at the end of computation and once per convolution. In importance space this re-normalization is achieved with subtraction:

\[
\begin{align*}
x_{pos} &= \begin{cases} x_{pos} - x_{neg} & \text{if } x_{pos} \geq x_{neg} \\ 0 & \text{if } x_{pos} < x_{neg} \end{cases} \\
x_{neg} &= \begin{cases} x_{neg} - x_{pos} & \text{if } x_{neg} \geq x_{pos} \\ 0 & \text{if } x_{neg} < x_{pos} \end{cases}
\end{align*}
\]

In delay space we re-normalize with the negative log difference exponential function nLDE (Equation 5), which is also symmetric about the \(y'\)-axis. The nLDE function is bounded by the inhibit function, similar to how the nLSE function is bounded by min. inhibit\((t_i, t_d)\) accepts two inputs, an inhibiting event set at time \(t_i\) and a data event arriving at time \(t_d\), and outputs an event at time \(t_d\) if and only if \(t_d < t_i\). When \(t_d \geq t_i\), no event will be output, which is equivalent to an event at time \(\infty\). We use min-of-inhibit functions to approximate nLDE:

\[
\text{min}(\text{inhibit}(x' + E_0, y' + F_0), \text{inhibit}(x' + E_1, y' + F_1), \ldots, \text{inhibit}(x' + E_{n-1}, y' + F_{n-1}))
\]

Using the same technique as the nLSE approximation, we simplify from two inputs to one, modeling min-of-inhibit approximations from Equation 7 where \(x' + y' = 0\) in Pyomo, and use KNITRO to optimize approximations with various numbers of inhibit-terms. Figure 5 shows a sample optimized nLDE approximation with four inhibit-terms.

2.2. Negative Numbers and Subtraction

Figure 4: Approximating nLSE with four max-terms. We only approximate positive \(x'\) values because Figure 3 is symmetric about the \(y'\)-axis.

Figure 5: Approximating nLDE with four inhibit-terms. nLDE’s shape is more difficult to approximate than nLSE’s shape, because nLDE converges to infinity as \(x'\) approaches 0, while nLSE converges to \(-\ln(2)\) as \(x'\) approaches 0 (see Figure 4).
A naive implementation of this can be seen in Figure 6 where we approximated the nLSE function (Equation 7).

This can be done by adding a constant \( K \) that is greater than or equal to the most negative approximation constant:

\[
nLSE(x', y') + K \\
\approx \min(x', y', \max(x' + C_0, y' + D_0), \ldots) + K \\
= \min(x' + K, y' + K, \max(x' + C_0 + K, y' + D_0 + K), \ldots)
\]

The now-positive max-term constants can now be implemented with constant delays, and the max and min functions can be replaced with LA and FA gates respectively. Finally, to handle swapping \( x' \) and \( y' \), as described in Section 2.1, we use a temporal comparator circuit [39] at the input to ensure proper ordering, preventing the need to double number of max-terms. A naive implementation of this can be seen in Figure 6 where each max-term has its own dedicated delay path.

However, this approach creates redundant delay, wasting both energy and area. With physical delay elements, energy consumption scales linearly with the magnitude of delay, so designs should minimize the number of redundant delays. This informs an optimized design shown in Figure 7, where there is only a single path of delay elements for each input. The max-term inputs are tapped from the proper location along the delay chain. Note that as max-terms are added, the \( C_i \) constants increase and the \( D_i \) constants decrease, so max-terms are connected in reverse order along their respective chains. This same shift-and-chain approach can be used to create a hardware implementation of the nLDE approximation function (Equation 7).

3. Recurrence Architecture

The accumulation operation in multiply-accumulate implies many connected summations. This can be achieved by all inputs arriving at the same time and being summed together. However, for most systems this is infeasible due to either the number of inputs or data naturally arriving over time. Therefore, a stateful system is required, which poses a problem for delay space. Temporal memories have been proposed in the past but they all rely on either emerging technologies or complex and sensitive analog circuits[24, 43, 27, 37]. To solve this problem we propose an alternative race logic approach that emulates a stateful system for efficient MACs.

This approach relies on the fact that in temporal systems, the arrival time of any event, \( t_x \), is meaningless without context. There must also be a reference time \( t_{ref} \), where the difference between the times \( t_x - t_{ref} \) represents \( x \)'s value. All inputs to race logic operations, including our nLSE approximations, must share the same reference time to function properly. In a system with fully parallelized inputs this is simple with a shared global reference time \( t_{ref} = t_{start} \). However, with serialized inputs each input will have a unique local reference time. Before the data can be processed with race logic it must be synchronized to the last input’s reference time. Inputs can be synchronized by adding delay equal to the difference between their local reference time and the last input’s reference time: \( t_x = t_x + (t_{start} - t_{local}) \) as shown in Figure 8a. These delay lines create state by holding in-flight data until all other inputs have arrived.

This represents an important trick where we can maintain the proper logical value of a signal with a shifting reference frame by adding a constant (delay). However, this requires as many delay lines as inputs, which wastes energy and area, making it infeasible for large systems. To avoid this, we propose an optimization that breaks apart the large nLSE block and operates on the inputs as they arrive, as shown in Figure 8b.

The equivalence of breaking apart the nLSE operations can be shown simply:

\[
- \ln(e^{-x} + e^{-y} + e^{-z}) = - \ln(e^{\ln(e^{-x} + e^{-y})} + e^{-z})
= - \ln(e^{-nLSE(x,y)} + e^{-z})
= nLSE(nLSE(x,y), z)
\]

This compute-upon-arrival approach also takes advantage of the shifted output of the nLSE hardware to offset some of the delay necessary to synchronize the frame of reference with the next input. Now the delay can be reduced from \( t_{cycle} \) to \( t_{cycle} - t_{nLSE} \). This also decreases the amount of in-flight data held by delay lines, while maintaining the same information utility.

This can be further optimized by taking nLSE’s output and the remaining frame shift delay and looping it back to the input as shown in the hollow arrows in Figure 8c. In order for this to operate properly, three constraints must be met:
The rise and fall time of the delay elements must be matched so that the integrity of the voltage pulse is maintained. The value of any input cannot be so large that it extends past the next value’s reference frame. A relaxation period must be introduced between cycles to ensure that the previous cycle’s falling edge does not interfere with the computation of the current cycle.

This recurrence takes advantage of reference shifting delay lines and causes the system to act like a classical state machine, so long as the inputs arrive at evenly spaced time intervals. This saves both energy and area by limiting the number and length of delay elements, as well as reducing the amount of nLSE hardware required. It’s worth noting that this approach is not restricted to a fully serial input. The two-input nLSE approximation units can be expanded to a tree of nLSE blocks with any number of inputs. Regardless of the size of the tree, the output can still be recurrent, creating a trade-off between the number of nLSE approximations, the area of the design, the minimum length of each cycle, and the number of cycles necessary for the computation.

### 4. Rolling Shutter Convolution Architecture

Most cameras rely on a technique called a rolling shutter [15] where individual columns or rows of pixels are captured and read out in parallel. This pipelines both the exposure time and the ADC readout times while causing the inputs to become available across fixed time intervals. This acts as a natural serialization for our recurrence architecture and can be leveraged for convolution. Even in systems without a rolling shutter, a staged readout is often applied to reduce the number of necessary ADCs.

![5-input nLSE](image)

![nLSE Approx](image)

![nLSE Approx](image)

Figure 8: Illustrations of different approaches to reference frame synchronization, where the y-axis indicates time and the x-axis indicates physical area added to the design. Figure (a) shows how each element can be individually delayed to ensure the proper reference frame. Figure (b) shows how computation can be performed as inputs arrive, and Figure (c) shows how the output of this computation can be looped to the same hardware (as shown by the hollow arrow) block with some delay to prevent hardware replication.

- The rise and fall time of the delay elements must be matched so that the integrity of the voltage pulse is maintained.
- The value of any input cannot be so large that it extends past the next value’s reference frame.
- A relaxation period must be introduced between cycles to ensure that the previous cycle’s falling edge does not interfere with the computation of the current cycle.

This recurrence takes advantage of reference shifting delay lines and causes the system to act like a classical state machine, so long as the inputs arrive at evenly spaced time intervals. This saves both energy and area by limiting the number and length of delay elements, as well as reducing the amount of nLSE hardware required. It’s worth noting that this approach is not restricted to a fully serial input. The two-input nLSE approximation units can be expanded to a tree of nLSE blocks with any number of inputs. Regardless of the size of the tree, the output can still be recurrent, creating a trade-off between the number of nLSE approximations, the area of the design, the minimum length of each cycle, and the number of cycles necessary for the computation.

4. Rolling Shutter Convolution Architecture

Most cameras rely on a technique called a rolling shutter [15] where individual columns or rows of pixels are captured and read out in parallel. This pipelines both the exposure time and the ADC readout times while causing the inputs to become available across fixed time intervals. This acts as a natural serialization for our recurrence architecture and can be leveraged for convolution. Even in systems without a rolling shutter, a staged readout is often applied to reduce the number of necessary ADCs.

![VDD Clk V pixel](image)

![VDD Clk V pixel](image)

Figure 9: (a) Demonstrates a simple starved inverter that forms the basis of voltage to time converters (VTC). (b) Shows the circuit diagram of an inverter-chain delay element with a transistor between the inverter output and ground to increase the delay of a single inverter.

4.1. Analog to Delay Space

Traditional camera systems rely on Analog-to-Digital Conversion (ADC) circuits to read pixel data prior to processing. However, low power systems have explored converting analog signals into the temporal domain [32, 34] for mixed signal processing. This approach has shown the potential for significant energy savings [14], and allows for temporal processing of data between the conversion to and from the time domain.

Many of these techniques attempt to achieve a linear mapping of input voltage to output timing [32, 12]. However, this is insufficient for our delay space computation as the negative log of the input is required. Instead, we require a voltage to time converter (VTC) that matches the negative log properties of delay space. This can be achieved by using a starved inverter, shown in Figure 9a, which is already at the core of many VTC systems [14].

The pixel voltage controls the current starving transistor, and the read clock acts as the input to the base inverter. As the pixel voltage increases, the delay of the inverter decreases, creating the value inversion necessary for delay space. Addi-
tionally, pixel voltage has a monotonically increasing impact on the delay of the clock signal, allowing it to approximate negative log for specific regions of interest. These two properties combined allow for the delay of the clock signal to be interpreted as a delay space value.

4.2. Delay Elements

Three previous approaches to delay elements for race logic have been proposed: discrete delay through chained DFFs [25], inverter chains, and starved inverters [26]. All approaches have their advantages, but the discretized approach would be impractical to integrate with the precise constant values required for the delay space approximations.

The starved inverter approach surrounds an inverter with two statically biased transistors [26]. These inverters act as current sources to reduce the rise and fall time of the output, resulting in controllable delays based on the bias voltages. However, this approach requires each starved inverter to be biased separately, introducing circuit complexity and additional energy consumption. Also, having a single large delay element maximizes the impact of random jitter (RJ) from a single element.

Conversely, an inverter chain uses a large number of identical inverters, with the effective load capacitance determining the delay from each individual inverter. In this approach the RJ for each element is independent and scales with the magnitude of delay, causing each additional element to reduce the overall impact of RJ [31]. In our design we use this approach and hard-code the load capacitance by adding a transistor between the output of each inverter and ground as shown in Figure 9b. The load capacitance is varied by changing the size of the ground transistor, allowing delay to be hard-coded into the hardware.

This chained inverter approach introduces an area and noise trade-off, as the larger the output capacitance, the fewer inverters are required for each chain. We evaluate how RJ impacts the accuracy of our approximations in section 5.2, which informs the design of the ground transistor.

4.3. Dedicated Convolution Engine

The core of this architecture is a hard-coded convolution MAC block shown in Figure 10 which handles all of the convolutions along a set of columns equal to the filter width. This block must be replicated for every filter application along the row axis of the given pixel array, given by

\[ \left\lceil \frac{\text{filterlength}}{\text{stride}} \right\rceil \]

Once the pixel values have been converted to delay space through the VTCs, each temporal pixel value is passed to all MAC blocks that utilize it. Figure 11 illustrates how this pixel data flows through the MAC block.

Each block has a static matrix of delay lines that includes every filter weight. As the corresponding pixels are processed by the VTCs they are passed to this matrix and the input can be distributed to any or all of the filter’s rows. The number of rows activated in a given cycle is given by

\[ \left\lceil \frac{\text{filterlength}}{\text{stride}} \right\rceil \]

where a stride of 1 indicates that every filter row will be used each cycle. ① Each activated filter row then performs element-wise delay, which functions as the multiplication of the MAC operation.

To fully utilize the outputs from the activated rows there must an accumulation unit for every potentially activated filter row. ② These accumulation units receive the results of the filter rows and subsequently sums the incoming values in delay space. The core of the accumulation unit is a tree of two-input nLSE approximation units performing delay space addition.
Whenever the tree is not fully symmetric, gaps in the tree must be path balanced by adding a delay equal to the inherent delay of the nLSE approximation hardware to maintain the proper reference frame. This delay is added as deep in the tree as possible to minimize the number of extra delay lines. When an image begins to be processed only one accumulation unit is activated, with the next one being activated every stride cycles. Eventually they will all be activated and operating in parallel, similar to a systolic array.

The result of this nLSE tree is then looped back as the recurrence discussed in Section 3. This loop must have a delay equal to the cycle time minus the inherent delay of the nLSE tree. Once an nLSE tree has received an input from each of the filter rows it passes its result to the output instead of being looped for recurrence. Then, in the next cycle the outputs of the filter rows are rotated so that each nLSE receives the next necessary input for their application of the filter. This ensures that there will be an output produced every stride cycles.

4.4. Split Value Representation

The split value representation presented in section 2.2 requires each MAC block to have multiple kernels to handle each combination of {positive, negative} {input, weight}. The temporal outputs of the VTCs are always non-negative, so we only need two kernels (positive input, positive weight and positive input, negative weight). If the inputs could be negative then we would need a total of four kernels to handle the two additional cases.

Since the weights are hard-coded, the filter weight matrix is split between the positive and negative sides. Weights that are zero in importance space, either because of the pos/neg split or the actual value is zero, become infinity. An infinite delay is the same as the path not existing, allowing the number of weight multiplications (delays) to stay constant with filter size, unaffected by the split value representation. Additionally, the number of nLSE approximations performed remains the same, the only extra operations are any required tree balancing and the recurrence delay lines.

Once a convolution have been completed, both positive and negative kernels must be routed to a delay space subtraction unit to re-normalize the values as described in Section 2.2. The subtraction unit evaluates an nLDE approximation (Section 2.3) on two inputs, and the result can be fed directly to the output. This output can either remain in delay space for further temporal computation, or be converted to the digital domain for traditional processing.

5. Evaluation and Results

5.1. Architectural Simulator

To explore the architectural space we created an architectural simulator that takes a system description as its input and produces a software representation of the architecture. The system description includes the image dimensions, the kernel shape, the number of kernels and the convolution stride. This architecture can be configured to change the unit scale, the maximum supply voltage swing, and the magnitude of each inverter’s delay (as a multiplier of the minimum inverter delay). We use the unit scale to indicate the connection between theoretical delay values and physical time values: for example an abstract delay of 1 could map to 5 ns.

These parameters are then used to estimate the area and energy consumption for the given system description. The energy estimates are based on SPICE simulations for delay lines using 65nm predictive technology models [47]. Area estimates are dependent on typical transistor sizes for 65nm nodes and an estimation for the total number of transistors in the system. We assume that the delay elements dominate both the energy and area and assume that the control logic is negligible. The architectural parameters are also used to implement noisy versions of delay and our approximations based on noise values from [31].

The generated architecture can be executed given an image data set and filter. The convolution is performed according to the compiled architecture with programmable multiplication, addition and subtraction functions. We use these programmable functions to ensure that when using importance space operations the architecture produces the exact same result as software convolution. We also verify that using exact delay, nLSE and nLDE provides the same result as software convolution when the results are converted from delay space to importance space.

5.2. Approximation Accuracy and Noise

To evaluate our approximation accuracy, we generate two uniform random values between zero and one, which correspond to positive values in delay space. The values are converted to delay space, the approximation is applied, and the result is then converted back to importance space. This approximation result is then compared to the exact operation being performed in importance space. We perform this operation a million times, then take the range-normalized RMS error to determine the overall accuracy.

We use this approach to evaluate how the number of approximation terms impacts the accuracy of our nLSE and nLDE approximations with infinite precision, shown in Figure 12a. The graph shows that additional approximation terms significantly increase the approximation accuracy until 7 or 8 terms where they start to provide diminishing returns.

However, when implemented in hardware, the accuracy of the approximation is also impacted by hardware timing noise. We use the noisy approximation simulator described in Section 5.1 to evaluate the two major sources of noise: power supply induced jitter (PSIJ) and random jitter (RJ) [31]. PSIJ is a product of the power delivery network (PDN) and will dominate the noise unless the swing in the supply voltage is carefully controlled. Figure 12b shows how the accuracy of our nLSE approximation suffers due to the noise introduced...
by varying $V_{DD}$ swings.

While this shows that PSIJ can significantly reduce the accuracy of the approximations, the low power nature of our convolution architecture puts less stress on the PDN, reducing potential voltage swings. If this is still insufficient to control the PSIJ, the voltage swing can be further controlled by adding decoupling capacitors to the PDN. RJ, on the other hand, cannot be controlled and is a function of the magnitude of each inverter’s delay [31]. However, since the RJ of each inverter is independent, the more inverters in a chain, the smaller the impact to the system. This creates a relationship between the delay of each inverter, the unit scale, and noise.

Figures 12c and 12d shows the impact of limited PSIJ (10mV $V_{DD}$ swing) and RJ for different unit scales with fixed delay element magnitudes. Figure 12c uses the smallest possible delay for each inverter, minimizing RJ, while Figure 12d uses an inverter with $50\times$ the minimal delay. Both of these show that there is a minimum unit scaling that must be met to ensure the max-terms can be utilized fully, otherwise the RJ noise dominates. With appropriate unit scaling and minimal noise there is little to no impact on the accuracy, but this requires significantly longer inverter chains to achieve the proper delay. However, Figure 12d shows that for a unit scale of 5ns the size of the inverter chains can be cut by $50\times$ with minimal noise impact. Beyond 5ns the hardware approximation improves slightly, but may not justify the extra energy required for the longer unit scaling.

Surprisingly the impact of noise, regardless of source, is larger as more approximation terms are added. Each additional term reduces the difference between approximation constants ($C_i$ and $D_i$ in Section 2.1), which increases the probability that the FA gate selects the wrong approximation term due to noise. The incorrect approximation term will then have a larger impact on the output than slight variations along the proper max-term.

Note that the nLDE approximation is also affected by noise, but because there is a larger difference between its approximation constants, the noise impacts the accuracy to a lesser degree. Due to space constraints we omit the nLDE noise trade-off graphs from this paper, but we consider this impact on nLDE accuracy in our architectural evaluation.

5.3. Architectural Evaluation

To investigate the relationship between approximation accuracy and the our convolution architecture, we run a design space exploration with our architectural simulator. For our exploration we use the Imagenette [16] dataset, which is a subset of the Imagenet [11] dataset, and scale each image to 150 by 150 pixels. Our architecture is then configured to run the Sobol function from OpenCV [2] which uses two $3\times3$ filters. We sweep the number of approximation terms for both nLSE and nLDE as well as the unit scale. The delay of each inverter is set to $50\times$ the minimal delay and the maximum $V_{DD}$ swing is set to 10mV.

For each configuration, the architectural simulator determines the energy consumption. Then it emulates all of the operations with appropriate noise in the same order as the simulated hardware for a single channel of the dataset. The Sobel convolution was calculated for five different images and
which leads to a larger energy consumption. Also, the energy consumption will marginally less energy than the Sobel function, this is largely due to the fact that its stride size is two, effectively halving the number of necessary computations and static hardware. GaussianBlur demonstrates that larger filter sizes consume significantly more resources, doubling the energy and area requirements from pyrDown while barely increasing the accuracy. However, because the height of the nLSE tree is the same for both functions, the minimum cycle time is the same.

We also compare our architecture to a state of the art pixel computation architecture [21] as shown in Table 3. We examine the same computation, four 4 × 4 filters with a stride of 2 for edge detection, as used by the convolution-in-pixel approach. For the race logic configurations use the same configuration as the architectural exploration with a 5ns unit scale, 10 max-terms and 20 inhibit-terms as our baseline. To ensure the same accuracy metric we normalized the RMS error to the range of our results to create a percent error.

For our energy per pixel per frame computation, we include the convolution energy cost plus the VTC conversion using values from [12] assuming the outputs are left in the temporal domain for further processing. We also include a comparison to a race logic architecture that requires the results to be converted to digital, using time-to-digital conversion energy from [13].

Our architecture uses less than an eighth of the energy per pixel for the convolution compared to the analog processing-in-pixel approach when the result can be left in the temporal domain for further processing. However, having to convert from the temporal to digital domain consumes more energy than the entire computation, reducing the energy savings of our approach. This motivates additional computation in the temporal domain, such as more convolutional layers or min/max selections. Also, our approach has no impact on the pixel array, allowing any convolution configuration or size. In-pixel processing requires dedicated configurations and it is difficult to achieve a stride of one. We also achieve slightly less error than the analog method, due to the noisy nature of analog
devices.

6. Related Work

Historically, work reducing the overheads of bit-parallel computation has centered around bit-serial computation. Recently this has been used to create energy efficient MACs [8], solving partial differential equations [33], and neural networks [19]. Memory cells have even been adapted to perform bit-serial computation near data [35, 19]. However, bit-serial computation replaces the bit-parallel computation with many cycles and high activity factors. It also requires registers and memory to keep track of data across cycles, increasing the area and energy consumption.

Similar to bit-serial computing, stochastic computing encodes information into a probabilistic bit-stream. Now computation can be done using just AND and OR gates [1]. However, random number generators are expensive, so researchers have investigated methods to reduce the dependence on independent streams [45]. This work has then shown how it can be applied in low power, near-sensor domains such as the brain [44]. However, this still requires a large number of cycles and expensive averaging circuitry.

Race logic has been proposed previously as a low power alternative to bit-serial and stochastic computing, and has shown to be very effective for dynamic programming [26] and decision trees [41]. However, our work is the first to propose an arithmetic framework for race logic, to the best of our knowledge. There exists another form of time-domain computation that uses pulse width to encode information [30]. General addition has been shown to be possible with time registers [38, 7], and multiplication has been shown by scaling this technique [37]. However, these operations work through iterative shifting and adding, requiring each input to fully complete before the next can begin. Also, these time registers have temporal limits before overflow issues must be handled.

Beyond alternative computing strategies, many analog hardware approaches have been coupled with image sensors. Convolution circuits have been placed alongside the photo diode for processing-in-pixel [20]. However, this approach has less accuracy than traditional approaches and causes the pixel size to increase significantly, reducing the resolution of cameras using this technique. Mixed near-sensor and in-sensor architectures have been proposed [23], but the near-sensor computation is based on conventional binary computation.

Race logic has been applied to 3D photon cameras [17] to reduce off-sensor bandwidth and computation. However, this is done using race logic to cleverly find the median without doing any actual arithmetic. Similarly, a time domain approach has been used in a retinal prosthesis [9] to perform energy efficient edge detection. They compare pulse widths with neighboring pixels and use mixed signal approaches to create a threshold for the differences. While this is similar to edge detection convolution (the Sobel operation), it cannot be generalized to other filters. We expect the new arithmetic capabilities of this energy efficient approach might open the door to even more applications in the future.

7. Conclusion

There is no question that convolution operations will continue to play an important role in sensor information processing, with applications including image processing, object detection, and neural networks. The power consumption of multiply-accumulate (MAC) operations is a key factor in convolutions integrated with sensor operation. Departing from the norm of performing complete analog-to-digital conversion for each pixel, we showed how to perform arithmetic on temporally encoded signals with remarkable energy efficiency.

At the heart of our approach is a negative log transformation, converting the traditional numeric space into a ‘delay space’. This mechanism enables multiplicative scaling by adding delays. We demonstrate the direct execution of negative log-space addition and subtraction in this new delay space, ensuring that normal associative and communicative properties of addition still apply in the transformed operations. Moreover, we show how strong approximations of these operations can be efficiently constructed from delay elements and existing CMOS logic elements.

Many computations execute iteratively, and to apply temporal techniques we need new techniques for chaining and recurrently operating these designs, multiplexing in both time and space. We show how time-division multiplexing aligns naturally with the staged ADC readout common to most modern sensor arrays. To establish the practicality of this approach, we present an automated transformation that carries traditional convolutions through to their delay space equivalents. This translation balances effort introduced by a new temporal equivalent of quantization and delay element noise. We use this approach to show how our approach can consume eight times less energy than another state-of-the-art convolution approach while achieving similar accuracy. We believe that this approach presents a powerful new set of design primitives with applications beyond convolution.

References


