Factoring Integers with a Brain-Inspired Computer

John V. Monaco and Manuel M. Vindiola

Abstract*—The bound to factor large integers is dominated by the computational effort to discover numbers that are B-smooth, i.e., integers whose largest prime factor does not exceed B. Smooth numbers are traditionally discovered by sieving a polynomial sequence, whereby the logarithmic sum of prime factors of each polynomial value is compared to a threshold. On a von Neumann architecture, this requires a large block of memory for the sieving interval and frequent memory updates, resulting in O(ln ln B) amortized time complexity to check each value for smoothness. This work presents a neuromorphic sieve that achieves a constant-time check for smoothness by reversing the roles of space and time from the von Neumann architecture and exploiting two characteristic properties of brain-inspired computation: massive parallelism and constant time synaptic integration. The effects on sieving performance of two common neuromorphic architectural constraints are examined: limited synaptic weight resolution, which forces the factor base to be quantized, and maximum indegree, which forces the factor base to be partitioned. Strategies for dealing with both constraints are introduced and the approach is validated by modifying `msieve`, which implements the multiple polynomial quadratic sieve, to use the IBM Neurosynaptic System (NS1e) as a coprocessor for integer factorization.

***Index Terms*—neuromorphic computing, quadratic sieve, smooth number, prime factorization**

I. Introduction

A number is said to be *smooth* if it is an integer composed entirely of small prime factors. Smooth numbers play a critical role in many interesting number theoretic and cryptography problems, such as integer factorization [1]. The presumed difficulty of factoring large composite integers relies on the difficultly of discovering many smooth numbers in a polynomial sequence, typically performed through a process called *sieving*. The detection and generation of smooth numbers remains an ongoing multidisciplinary area of research which has seen both algorithmic and implementation advances in recent years [2].

This work demonstrates how current and near future neuromorphic architectures can be used to efficiently detect smooth numbers in a polynomial sequence. The *neuromorphic sieve* exploits two characteristic properties of neuromorphic computing architectures inspired by the functioning of the brain:

- *Massive parallelism*: the mammalian brain contains billions of neurons that operate concurrently. Like the brain, neuromorphic architectures contain many neuron-like units which are updated in parallel, in stark contrast to conventional von Neumann architectures which rely on a central processing unit (CPU).
- *Constant-time synaptic integration*: a single neuron in the brain may receive electrical potential inputs along synaptic connections from thousands of other neurons. The incoming potentials are continuously and instantaneously integrated to compute the neuron's membrane potential. Like the brain, neuromorphic architectures aim to perform synaptic integration in constant time, typically by leveraging physical properties of the underlying device.

Unlike the traditional CPU-based sieve, the factor base is represented in space (as spiking neurons) and the sieving interval in time (as successive time steps). Sieving is performed by a population of leaky integrate-and-fire (LIF) neurons whose dynamics are simple enough to be implemented on a range of current and future architectures. Integer factorization is achieved using a neural processing unit (NPU) for the sieving stage, alongside a CPU host.

Sieving can be thought of as a binary classification of smooth/non-smooth numbers whereby a tradeoff between sensitivity and specificity is achieved by varying a detection threshold. In this light, we consider how some architectural constraints affect sieving performance. Neuromorphic architectures are subject to such constraints as low-resolution synaptic weights, realized through limited bit size or weight sharing, and maximum indegree, which specifies a limit on the number of inputs to any single neuron. In terms of sieving, low-resolution weights force the factor base to be quantized and maximum indegree forces the factor base to be partitioned. Four strategies for dealing with each of these constraints are described and evaluated in order to better understand the problem size a particular architecture can be applied to and, conversely, what type of architecture would be needed for a given problem size. The approach is validated by modifying `msieve`, one of the fastest publicly available integer factorization implementations, to use the IBM Neurosynaptic System (NS1e) as a coprocessor for the sieving stage.

This article is an extension of earlier work in which the neuromorphic sieve was introduced and synaptic weight quantization strategies described [3]. Building on this previous effort, the contributions of this article include:

1) The treatment of sieving as a binary classification and a formula for choosing a detection threshold in order to maximize CPU utilization and minimize sieving time.
2) Strategies for dealing with neuromorphic architectures that impose a maximum indegree.
3) Experimental results obtained for integers up to 128 bits using the NS1e as a sieving coprocessor.
4) Complexity analysis and estimated sieving performance on near-future neuromorphic architectures.

The rest of this article is organized as follows. Section II reviews background material related to integer factorization,

Manuscript received June 11, 2017; revised September 19, 2017; accepted October 30, 2017.

The authors are with the U.S. Army Research Laboratory, Aberdeen Proving Ground, MD 21005, USA (e-mail: john.v.monaco2.civ@mail.mil, manuel.m.vindiola.civ@mail.mil).

smooth number detection, and spiking neural networks. Section III describes the neuromorphic sieve. Sieving as a binary classification is introduced in Section IV, wherein the problem of choosing a threshold is described. Architectural constraints are described in Section V along with strategies for dealing with the constraints. The implementation of the neuromorphic sieve on the NS1e and experimental results are in Section VI. Section VII further examines the results through simulation and contains estimated sieving times on near-future neuromorphic devices. Finally, Section VIII contains a discussion, and conclusions are made in Section IX.

II. Background

A. Integer Factorization

Integer factorization is presumed to be a difficult task when the number to be factored is the product of two large primes. Such a number $n = pq$ is said to be a *semiprime*[1] for primes p and q, $p \neq q$. The RSA algorithm relies on the difficulty of factoring n when p and q are of the same magnitude (close to the square root of n) [4]. As $\log_2 n$ grows, i.e., the number of bits to represent n, the computational effort to factor n by trial division grows exponentially.

Dixon's factorization method attempts to construct a congruence of squares, $x^2 \equiv y^2 \mod n$ [5]. If such a congruence is found, and $x \not\equiv \pm y \mod n$, then $\gcd(x-y, n)$ must be a nontrivial factor of n. A class of subexponential factoring algorithms, including the quadratic sieve, build on Dixon's method by specifying how to construct the congruence of squares through a linear combination of smooth numbers [6].

Given smoothness bound B, a number is B-smooth if it does not contain any prime factors greater than B. Additionally, let $\mathbf{v} = \left[e_1, e_2, \ldots, e_{\pi(B)}\right]$ be the exponents vector of a smooth number s, where $s = \prod_{1 \leq i \leq \pi(B)} p_i^{v_i}$, p_i is the ith prime, and $\pi(B)$ is the number of primes not greater than B. With a set of $\pi(B)+1$ unique smooth numbers $\mathcal{S} = \left\{s_1, s_2, \ldots, s_{\pi(B)+1}\right\}$, a perfect square can be formed through some linear combination of the elements of $\mathcal{S}$, $y^2 = \prod_{s_i \in S} s_i$. The reason for this is that there exists at least one linear dependency among a subset of the $\pi(B)+1$ exponents vectors that contain $\pi(B)$ elements each. Gaussian elimination or block Lanczos algorithm can be used to uncover this linear dependency [7].

Smooth numbers are detected by sieving a polynomial sequence. Sieving relies on the fact that for each prime p, if $p \mid f(x)$ then $p \mid f(x+ip)$ for any integer i and polynomial f. To sieve the values $f(x)$ for $0 \leq x < M$ on a von Neumann architecture, a length M array is initialized to all zeros. For each polynomial root r of each prime p in the factor base, $\ln p$ is added to array locations $r + ip$ for $i = 0, 1, \ldots, \frac{M}{p}$. This step is typically performed using low precision arithmetic, such as with integer approximations to $\ln p$. After looping through each prime in the factor base, array values above a certain threshold τ will correspond to polynomial values that are smooth with high probability[2]. This process is referred to hereafter as CPU-based sieving.

[1]Not to be confused with *pseudoprime*, which is a probable prime.

[2]By exploiting the fact that $\ln ab = \ln a + \ln b$.

Since the actual factorizations are lost after sieving, the smooth candidates must subsequently be factored over the factor base $\mathcal{F}$, which also serves as a definite check for smoothness. Factoring a number with small prime factors can be done efficiently, and this effort can be neglected as long as there are not too many false positives [8].

The quadratic sieve [6] detects smooth numbers of the form

$$f(x) = \left(x + \lceil\sqrt{n}\rceil\right)^2 - n \tag{1}$$

where $x = -\frac{M}{2}, \ldots, \frac{M}{2}-1$. The factor base $\mathcal{F}$ contains primes p up to B such that n is a quadratic residue modulo p, i.e., $r^2 \equiv n \mod p$ for some integer r. This ensures that each prime in the factor base (with the exception of 2) has two modular roots to the equation $f(x) \equiv 0 \mod p$, increasing the probability that $p \mid f(x)$. If $\mathcal{F}$ contains b primes, then at least $b+1$ smooth numbers are needed to form the congruence of squares.

It is the sieving stage of the quadratic sieve that is the focus of this work. Sieving comprises the bulk of the computational effort in the quadratic sieve and the relatively more complex number field sieve (NFS) [9]. On a von Neumann architecture, sieving requires at least $\frac{1}{2}M + M\sum_{p \in \mathcal{F} \setminus 2} \frac{2}{p}$ memory updates where $\mathcal{F} \setminus 2$ is the set of factor base primes excluding 2. Given probability u^{-u} of any single polynomial value being smooth, where $u = \frac{\ln n}{2 \ln B}$, an optimal choice of B is $\exp\left(\frac{1}{2}\sqrt{\ln n \ln \ln n}\right)$ [8]. This yields a total runtime of B^2, where the CPU-based amortized time to sieve each value in the interval is $\ln \ln B$.

B. Spiking Neural Networks

Neuromorphic computing encompasses several fields of research geared towards the design, development, and implementation of computers inspired by the the biological brain. In stark contrast to the von Neumann architecture, comprised of a central processing unit (CPU) and a separate memory unit, neuromorphic computing architectures make no distinction between processor and memory as memory is co-located with neuron-like processing units [10]. This feature promises to alleviate the von Neumann bottleneck between memory and CPU [11], and, faced with the eventual demise of Moore's Law, a wide range of neuromorphic architectures have recently emerged [12]. Many of these architectures implement or simulate spiking neural networks.

Spiking neural networks are considered the third generation of neural networks, coming after the multilayer perceptron (MLP) and "activation function" networks [13]. The spiking neuron model aims to provide more biological realism than its predecessors, whereby the MLP can be viewed as a snapshot in time of a SNN, and activation function networks can be viewed as encoding spike rates over time.

There are many flavors of spiking neuron model which vary by number of parameters, neuron dynamics, and response properties. This work uses a discrete leaky integrate-and-fire (LIF) neuron, considered to be one of the simpler models and applicable to a wide range of neuromorphic architectures [12]. Let $V(t)$ and $A(t)$ be the membrane potential and spike output, respectively, at time t. The LIF neuron model used in this work is defined by the following update equations:

1) Integrate. Spikes emitted by presynaptic neuron i at time $t-1$, denoted by $A_i(t-1)$, are received at time t over weighted synaptic connections w_i. The weighted spikes are integrated and a leak λ is applied. The neuron's membrane potential is updated according to

$$V(t) = V(t-1) + \sum_i A_i(t-1)\, w_i + \lambda \,. \tag{2}$$

2) Spike. The neuron emits a spike if it's membrane potential has reached the threshold α,

$$A(t) = \begin{cases} 1 & \text{if } V(t) \geq \alpha \\ 0 & \text{otherwise} \end{cases} \,. \tag{3}$$

3) Reset. If the neuron spikes, it's membrane potential resets to 0. The membrane potential is also clamped by a lower bound of 0. The reset behavior is given by

$$V(t) = \begin{cases} 0 & \text{if } V(t) \geq \alpha \\ 0 & \text{if } V(t) < 0 \\ V(t) & \text{otherwise} \end{cases} \,. \tag{4}$$

Time advances in discrete steps, and Equations 2, 3, and 4 are applied in succession at each time step. Additionally, the initial membrane potential of each neuron can be specified at time $t = 0$, denoted by V_0. The complete set of parameters for each neuron is $\{V_0, \alpha, \lambda\}$ and w_i specify the weighted synaptic connections.

III. Neuromorphic Sieve

A. Network Construction

Construction of the neuromorphic sieve is demonstrated through an example using the semiprime $n = 91$, quadratic polynomial $f(x) = \left(x + \lceil\sqrt{91}\rceil\right)^2 - 91$, and sieving interval $x = -5, \ldots, 4$. The smoothness bound is set to $B = 5$ resulting in factor base $\mathcal{F} = \{2, 3, 5\}$, the primes up to B such that the Legendre symbol $\left(\frac{n}{p}\right) = 1$, i.e., n is a quadratic residue modulo p. Sieving is also performed with prime powers, p^e for $e > 1$, that do not exceed the magnitude of the polynomial over the sieving interval, namely 3^2, 3^3, and 5^2. Powers of 2 are not included since they do not have any modular roots to the equation $f(x) \equiv 0 \mod 2^e$ for $e > 1$.

The neuromorphic sieve represents the factor base in space, as tonic spiking neurons, and the sieving interval in time through a one-to-one correspondence between time and the polynomial sequence. Let $t \equiv (x - x_{\min})$, where t is time, x is the sieving location that corresponds to t, and x_{min} is the first sieving value. Then, polynomial values can be calculated by $f(x) = f(t + x_{\min})$. Each tonic neuron corresponds to a prime (or a power of a prime) in the factor base and spikes only when it divides the current polynomial value. *If enough tonic neurons spike at time t, then $f(x)$ is likely smooth since each neuron represents a factor of $f(x)$.* This formulation reverses the roles of space and time from the CPU-based sieve, in which the sieving interval is represented in space, as an array, and a doubly nested loop iterates over over primes in the factor base and locations in the sieving interval.

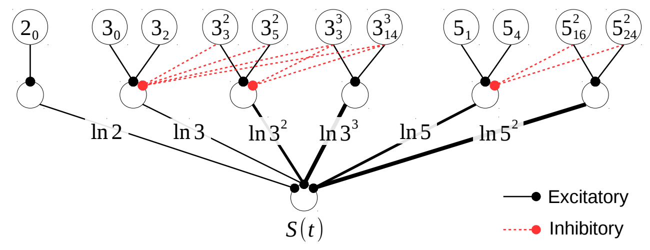


Fig. 1. Example neuromorphic sieving network. Top layer contains tonic spiking neurons; middle layer selects the highest prime power; bottom layer performs the test for smoothness. Reprinted from [3].

The sieving network is comprised of three layers of spiking neurons arranged in a tree structure (Figure 1). The top layer contains tonic spiking neurons for each root of each prime in the factor base, as well as prime powers up to the magnitude of the polynomial. The middle layer selects the highest power of each prime that divides the polynomial value by inhibiting the lower powers. The bottom layer contains a single neuron that performs a test for smoothness by integrating the log-weighted factors. Configuration of each layer is as follows.

Top layer. For each modular root r of each prime power p^e (including factor base primes, for which $e = 1$), designate a neuron that will spike with period p^e and phase r (Figure 1, top layer, given by p^e_r, where subscripts denote the modular root). Due to the equivalence between t and $f(x)$, this neuron will spike only when $p^e | f(x)$. It is also the case that if $p^e | f(x)$ then $p^e | f(x + ip^e)$ for any integer i, thus only tonic spiking behavior for each modular root is required. Since the factor base is comprised only of primes up to B such that n is a quadratic residue modulo p, each prime $p > 2$ in the factor base has exactly two modular roots. Generally, a prime may have up to d modular roots to the equation $f(x) \equiv 0 \mod p$ for a polynomial of degree d.

Middle layer. The tonic neurons for each prime (top layer) are connected through excitatory synapses ($w = 1$) to factor neurons that spike if either modular root spikes (Figure 1, middle layer). Therefore, the respective factor neuron spikes if $p^e_{r_1} | f(x)$ or $p^e_{r_2} | f(x)$. Using a LIF neuron model, this behavior is achieved using synapse weights of the same magnitude as the postsynaptic neuron threshold. Inhibitory connections ($w = -1$) are then created between the prime powers in the top layer and lower prime powers in the middle layer so that higher powers suppress the spiking activity of the lower powers. The inhibitory connections are required to ensure that a spike is generated only for the highest prime power that divides the current polynomial value.

Bottom layer. Synapse connections from the factor neurons (middle layer) to the smoothness neuron are weighted proportional to $\ln p^e$ (Figure 1, bottom layer). The smoothness neuron emits a spike if the logarithmic sum of factors reaches the threshold $\ln |f(x)|$, signaling whether $f(x)$ can be completely factored over the factor neurons that spiked in the middle layer. This stateless behavior is achieved by setting the leak $\lambda = -\ln |f(x)|$ and threshold $\alpha = 0$.

TABLE I
NEURON PARAMETERS FOR THE THREE NEURON TYPES.

Neuron Parameter	Tonic (top)	Factor (middle)	Smoothness (bottom)
Potential V_0	$-(r+1-x_{min}) \mod p^e$	0	0
Threshold α	p^e	1	0
Leak λ	1	0	$-\ln(x)$

The neuron parameters for each layer are summarized in Table I (see Section II-B for neuron model). The middle and bottom layers are stateless, i.e., their membrane potentials never carry over to the next time step, while the dynamics in the top layer encode successive polynomial values.

B. Space and Time Complexity

In general, a neuromorphic sieving network with factor base of size b primes requires at least $3b+1$ neurons ($2b$ in the top layer, b in the middle, and 1 in the bottom) and $3b$ excitatory synaptic connections. This does not include neurons for prime powers and inhibitory synaptic connections, which depend on the magnitude of the polynomial $|f(x)|$ and sieving interval length M. Prime powers with roots outside of the sieving interval need not be considered since these will never spike.

Given the probability that any single polynomial value $f(x)$ is B-smooth is u^{-u}, where $u = \frac{\ln n}{2\ln B}$, the expected number of sieve values that must be examined to find one smooth number is u^u [8]. Since the factor base contains b primes, at least $b+1$ smooth numbers are needed to uncover a linear dependency among the exponents vectors and complete the factorization. With the amount of work per sieve value being $\ln\ln B$ on a von Neumann architecture, the total CPU-based sieving time is given by

$$T(n) = u^u(b+1)\ln\ln B \tag{5}$$

where $B = \exp\left(\frac{1}{2}\sqrt{\ln n \ln\ln n}\right)$. Comparatively, the neuromorphic sieve spends exactly one step per sieve value, which eliminates the $\ln\ln B$ term above, leading to a total sieving time of

$$T(n) = u^u(b+1) \ . \tag{6}$$

Table II shows the time complexity of the CPU-based sieve and the neuromorphic sieve as a function of n bits. The largest relative gains are achieved for small problems since the speedup itself follows $\ln\ln B$. For example, a 3× speedup is achieved around 512 bits, and this grows to only a 3.5× speedup at 1024 bits. However, there is a hidden cost in Equation 5 which is often ignored: that is the smooth number detection rate, i.e., the probability of detecting a number that is actually smooth. In sieving time complexity analyses, the effects of missed smooth numbers are often considered to be negligible on total runtime. However, there is a tradeoff between an algorithm's ability to correctly detect numbers that are smooth and correctly reject numbers that are not smooth, due to such optimizations as low precision arithmetic, discarding small primes in the factor base, and ignoring prime powers. In the next section, we examine this tradeoff by treating smooth number detection as a binary classification problem and varying the choice of threshold.

TABLE II
CPU VS NPU SIEVING TIME COMPLEXITY.

n bits	CPU Time	NPU Time	Speedup
64	6.4×10^{3}	3.4×10^{3}	1.87
128	1.9×10^{6}	0.8×10^{6}	2.30
256	1.1×10^{10}	0.4×10^{10}	2.72
512	4.7×10^{15}	1.5×10^{15}	3.13
768	1.6×10^{20}	0.5×10^{20}	3.36
1024	1.3×10^{24}	0.4×10^{24}	3.53

IV. Sieving as a Binary Classification

Sieving a polynomial sequence can be thought of as a binary classification problem in which smooth integers are the positive class and non-smooth integers are the negative class. With the neuromorphic sieve, each polynomial value is assigned a score that is the membrane potential of the smoothness neuron (Figure 1, bottom layer). As the threshold of the smoothness neuron is varied, there is a tradeoff between false positive rate (FPR) and false negative rate (FNR) from which a receiver operating characteristic (ROC) curve is obtained. This same principle applies to CPU-based sieving: due to errors resulting from low precision arithmetic, some of the array values above the threshold will end up not being smooth and some values that are smooth will remain below the threshold, with the threshold controlling a tradeoff between FPR and FNR.

Ideally, the detection threshold of the smoothness neuron[3] would follow the logarithmic magnitude of the polynomial such that $\lambda(t) = \ln|f(t+x_{\min})|$. Typically, this cannot be realized since the leak in most neuromorphic architectures that implement a LIF neuron model is constant, i.e., $\lambda(t) = \lambda$. Due to this constraint, and other architectural constraints described in the next section, both the CPU-based sieve and neuromorphic sieve must operate somewhere along the ROC curve, trading more false positives for a better chance of not missing any smooth values. As the threshold increases, the number of false positives will decrease. Conversely, the number of false negatives increases with the threshold, and as a result, more time must be spent sieving due to lower sensitivity. Modifying Equation 6 to account for the smooth detection rate, the total sieving time becomes

$$T(n) = \frac{u^u(b+1)}{1-\mathrm{FNR}} \ . \tag{7}$$

From Equation 7, it can be seen that sieving time is minimized with a perfect smooth detection rate ($\mathrm{FNR} = 0$), the case that is traditionally assumed in sieving time complexity analyses.

A. ROC Curve Example

As an example, consider the sieving performed for a single 64-bit integer, $n = 12425785886937261613 = 3973371161 \times 3127265333$. A smoothness bound of 799 is chosen, resulting in 58 factor base primes. Including prime powers, the factor base contains 157 factors. There are 76 smooths over a sieving

[3]Note that the leak λ is used as a detection threshold to achieve stateless behavior in the smoothness neuron. See Section II-B for the neuron model and Table I for neuron parameters.

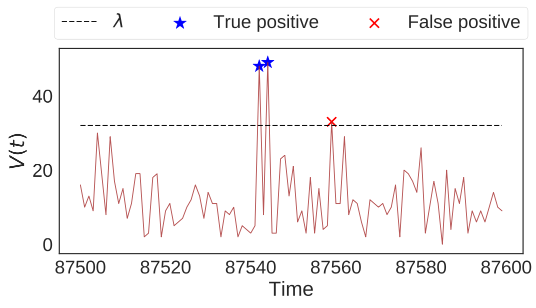
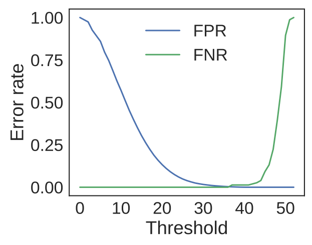
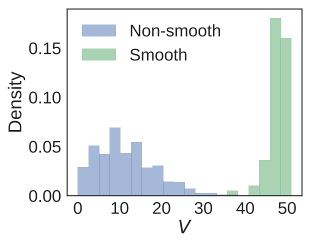


Fig. 2. Membrane potential of the smoothness neuron during time $t = [87500, 87599]$. The smoothness neuron spikes three times; of these, two are actually smooth (true positives) and one is not smooth (false positive).



(a) Error rates



(b) Score densities.

Fig. 3. Error rates and score densities obtained by the neuromorphic sieve for the 64-bit semiprime $n = 12425785886937261613$. Note that the FNR and FPR in (a) can be interpreted as the cumulative distribution (CD) and complementary CD, respectively, of the score densities shown in (b).

interval of length 2^{17}, which is centered on 0. Sieving starts with $x = 1 - 2^{16}$ at time $t = 0$ and ends with $x = 2^{16}$ at time $t = 2^{17}$.

The detection threshold of the smoothness neuron is set to $\lambda = 32$, just below the mean log magnitude of the polynomial, which is about 33. At this point, the FPR is 1.033% and the FNR is 0%. Figure 2 shows a portion of the sieving interval during time $t = [87500, 87599]$ (chosen to include both true positives and false positives). During this interval, three values are detected as smooth. Of these, 2 are actually smooth (true positives) and 1 is not smooth (false positive). There are no false negatives in this interval.

The complete ROC curve is derived by varying the threshold of the smoothness neuron and calculating the FPR and FNR at each threshold value. Figures 3a and 3b show the FPR and FNR as a function of threshold value and the score densities of each class, respectively. The equal error rate (EER) is the point on the ROC at which FPR and FNR are equal, in this case 0.165%.

B. Choosing a Threshold

The neuromorphic sieve operates alongside a CPU host, which must check each candidate for smoothness and recover the prime factorizations over $\mathcal{F}$ needed for the linear algebra stage. B-smooth candidates are generated by the NPU and transmitted to the CPU host as spike timings. Due to the correspondence between time and polynomial value, the CPU host needs only to know the time at which the NPU emits a spike to recover the polynomial value and check the candidate for B-smoothness.

Given the speed of the NPU, however, the choice of threshold on the smoothness neuron determines how often the candidates will be generated, and the magnitude of the polynomials and factor base size will determine how long it takes the CPU to check each candidate. The optimal choice of threshold is one in which CPU utilization is maximized, but does not exceed 100%. This ensures that the NPU generates the maximum number of candidates that the CPU host can handle, but not too many candidates are generated such that they must be buffered. In other words, the threshold should be the minimum choice that does not exceed a 100% CPU utilization rate, thereby minimizing the FNR.

Under the assumption that smooth candidates follow a random (Poisson) process, and the time to check each candidate is constant, neuromorphic sieving can be treated as an M/D/1 queue. Let t_c be the time to check each candidate for smoothness, P_s be the spiking probability of the smoothness neuron, and f the clock frequency of the NPU. The arrival rate is then given by the spiking rate of the smoothness neuron, fP_s, and the service rate is given by the inverse time it takes the CPU to check each candidate for B-smoothness, $\frac{1}{t_c}$. From this, CPU utilization is calculated as

$$\rho = fP_s t_c \, . \tag{8}$$

The threshold λ of the smoothness neuron should be adjusted such that ρ is just below 1. Otherwise the arrival rate will exceed the service rate, resulting in an unbounded buffer and more time spent checking candidates for smoothness by the CPU host than the actual sieving.

In the example above, consider an NPU with clock rate $f = 1\,\text{GHz}$ and a CPU host check time of $t_c = 1\,\mu\text{s}$. Then, the threshold should be chosen where $P_s = \frac{10^6}{10^9} = 0.001$. This occurs at $\lambda = 39$ which gives an FPR of 0.056% and FNR of 1.316%. Alternatively, Equation 8 can be used to determine the optimal clock rate of the NPU, given the CPU check time and desired spiking probability. Let the desired spiking probability be $P_s = 0.1$, which achieves empirical FPR and FNR of 9.093% and 0%, respectively, at $\lambda = 22$. Then, $f = \frac{10^6}{0.1} = 10\,\text{MHz}$ is the maximum NPU clock rate that can be used without overwhelming the CPU.

V. Architectural Constraints

Neuromorphic hardware may impose architectural constraints that limit the performance of the neuromorphic sieve or, alternatively, limit the problem size (n bits) that a particular architecture can be applied to. Accuracy, speed, and energy efficiency are often conflicting requirements for neuromorphic chip design, which must typically trade one for the other. Low-resolution synaptic weights may be employed for increased speed at the cost of decreased accuracy for some tasks. The number of synaptic connections to any single neuron may also be limited (creating a maximum indegree) for decreased communication requirements. This section addresses two such architectural constraints in neuromorphic systems and describes strategies for mitigating the performance degradation imposed by the constraints.

A. Limited Resolution: Synaptic Weight Quantization

Synaptic weight resolution refers to the number of different values a synaptic connection weight can assume. On low-resolution neuromorphic architectures, synapse weights may be restricted to, e.g., 8-bit values which reduces the amount of space and energy required to compute compared to higher-resolution representations, such as 32 bits or 64 bits. In the worst case, synaptic connections are binary valued and can be either "on" or "off" (alternatively, -1 or +1). Such a binary neural network can be implemented efficiently in hardware since every synaptic connection has the same strength, and synaptic integration amounts simply to counting the number of spikes received at each neuron which effectively eliminates multiplications. Recent work in machine learning has demonstrated that this is not especially detrimental to performance, and state-of-the-art performance can be achieved even with networks of only binary precision for some tasks [14].

Limited synaptic weight resolution also comes through the use of weight sharing whereby a population of synaptic connections must share a smaller number of unique weights. For example, on TrueNorth, synapse weights are stored in signed 9-bit memory locations, permitting up to 2^9-1 different values ranging from $[-255, 255]$; however each neuron can receive spikes from at most 4 unique synapse weights. This leads to an efficient digital implementation, since synaptic integration can be performed using a lookup table and just 4 multiplications [15].

To summarize, *bit size* limits the number of unique values a single synapse can assume, and *weight sharing* limits the number of unique values a population of synapses can assume. Both bit size and weight sharing limit synaptic weight resolution and require the synapse weights of the neuromorphic sieve to be quantized. Specifically, given a factor base of size F (including prime powers) and an architecture that has k_w unique weights, with $F > k_w$, the synapse weights from the factor neurons to the smoothness neuron (middle-bottom layers, Figure 1) must be quantized to k_w unique values. To accomplish this, four different weight quantization strategies are described.

Regress: With k_w weights shared among F factors, the regress strategy attempts to construct a step function with k_w steps that minimizes the mean squared error (MSE) to the log factors. This is performed by fitting a single variable regression tree with k_w leaf nodes to the log factors using MSE as the optimization criterion. The quantized weight for each factor is determined through a lookup in the regression tree.

Inverse: Similar to the regress strategy, the inverse strategy uses a step function to obtain the quantized weights. Instead of the MSE, the objective function is a weighted MSE with weights given by the inverse logarithm of each factor. This coerces the quantized weights of smaller, frequent factors to be more accurate than larger factors.

Uniform: The uniform strategy assigns each factor a weight of 1; thus the smoothness neuron simply counts the number of factors that divide any sieve value. This strategy is compatible with architectures that implement binary networks.

Integer: The integer strategy assigns each factor a weight equal to the log factor rounded to the nearest 32-bit integer, compatible only with architectures that support 32-bit weights with no weight sharing.

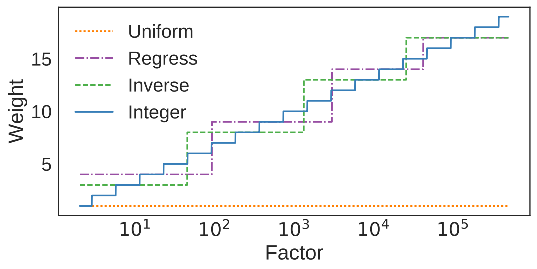


Fig. 4. Synaptic weight quantization example using $k_w = 4$ shared weights with the exception of the integer strategy which uses 32-bit weights and is not compatible with weight sharing. Reprinted from [3].

The four quantization strategies are summarized in Figure 4 for $k_w = 4$ applied to the sieving example in Section IV. Using only integer arithmetic, the integer strategy is optimal as it most closely approximates the log function. The uniform strategy is a worst case in which only binary (0 or 1) weights are available. Note that only the regression, inverse, and uniform strategies, which have at most 4 unique weights, are able to run on an architecture with shared weights, such as TrueNorth. The integer strategy exceeds the limit of 4 unique weights to any single neuron, thus is not compatible with architectures that have shared weights.

B. Limited Topology: Factor Base Partitioning

A key architectural feature that the neuromorphic sieve exploits is the ability to perform a summation in $O(1)$ time through synaptic integration. This occurs at the smoothness neuron where the spikes from all of the factor neurons are integrated to perform a logarithmic summation of the factors. However, not all neuromorphic processors have the ability to integrate spikes from an unlimited number of presynaptic neurons in $O(1)$ time. In many cases, there is a maximum indegree that limits the number of inputs that can be integrated in a single step. For example, on TrueNorth, each neuron has 256 dendrites along which it can receive spikes and, alternatively, can transmit spikes to up to 256 axons, making the maximum indegree/outdegree of each neuron 256 [15]. Generally, having a maximum indegree permits each neuron to perform a summation of up to k_d inputs in $O(1)$ time, where k_d is the maximum indegree of the architecture. This limits the number of unique factors that can be integrated in a single step to k_d, and as a result, a factor base with more than k_d elements must be partitioned into k_d subsets.

As an example, consider the sieving network from Section III with $F = 6$ (including prime powers) and suppose this problem must be implemented on a neuromorphic architecture with $k_d = 3$. That is, the factor base must be partitioned into 3 subsets. This is performed by creating an additional *partition layer* with exactly k_d neurons between the factor neuron layer and the smoothness neuron.

Figure 5 demonstrates a partitioned factor base and depicts the network activity at time $t = 6$ in which factors 2, 3, and 5 are active (shown in gray). The factor neurons are partitioned

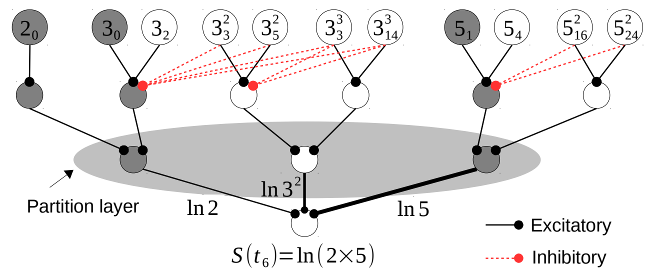


Fig. 5. Neuromorphic sieve example with maximum indegree of $k_d = 3$. The partition layer splits the factor base neurons into k_d subsets to satisfy the architectural constraint having maximum indegree k_d (compare to Figure 1). This example shows neuron activity at time $t = 6$ using the example from Section III; factors $\{2, 3, 5\}$ are active and a collision occurs at the leftmost partition neuron (3rd layer). A partitioning strategy amounts to forming the connections between the factor layer and partition layer (2nd-3rd layers).

Factors	2	3	5	3^2	5^2	3^3
Partitioning Strategy						
Constant	$\{1$	$1\}$	$\{1$	$1\}$	$\{1$	$1\}$
Logarithmic	$\{\ln 2$	$\ln 3$	$\ln 5\}$	$\{\ln 3^2$	$\ln 5^2\}$	$\{\ln 3^3\}$
Inverse Log	$\{\frac{1}{\ln 2}\}$	$\{\frac{1}{\ln 3}$	$\frac{1}{\ln 5}\}$	$\{\frac{1}{\ln 3^2}$	$\frac{1}{\ln 5^2}$	$\frac{1}{\ln 3^3}\}$
Probabilistic	$\{\frac{1}{2}\}$	$\{\frac{2}{3}\}$	$\{\frac{2}{5}$	$\frac{2}{3^2}$	$\frac{2}{5^2}$	$\frac{2}{3^3}\}$

Fig. 6. Four partitioning strategies applied to factor base $\mathcal{F} = \{2, 3, 5, 3^2, 5^2, 3^3\}$ with $k_d = 3$. Each strategy is defined by its partition function $g(p^e)$. Subsets are formed by applying the partition function to the factors and then creating subsets of equal sum using Algorithm 1.

into k_d subsets with the factors from each subset connected to a single partition neuron. The partition neuron spikes if *any* of the factor neurons spike, effectively performing a logical OR operation between the inputs. The connections between the partition layer and the smoothness neuron are weighted by the smallest log factor in each subset, as this is the minimum factor represented by each partition neuron.

1) Spike Collisions: A *spike collision* occurs when spikes from two different factors in the same subset are received by a neuron in the partition layer at the same time. When this happens, the partition neuron will activate and send a spike to the smoothness neuron; however, the smoothness neuron knows only that at least one factor in the subset has spiked. In Figure 5, the factor neurons are partitioned into k_d subsets of similar size and the spikes from neurons 2 and 3 collide since they are both connected to the same neuron. The membrane potential of the smoothness neuron is $S(6) = \ln(2 \times 5)$ since it sees only spikes along synapses with weights $\ln 2$ and $\ln 5$ corresponding to the subsets with minimum factors 2 and 5, respectively; it is absent to the fact that the 3 has also spiked.

The probability of a spike collision can be determined analytically. This is the probability of at least m out of N events occurring where $m = 2$ and N is the number of possible events [16, Ch. IV, Theorem 3.1]. Applied to spike collisions, the event probabilities are given by the probability that each factor neuron spikes, which is $\frac{2}{p^e}$ for factors with 2 modular roots. The probability of each partition neuron receiving at least m spikes is given by

$$\begin{aligned} P_m &= S_m - \binom{m}{m-1} S_{m+1} + \\ &+ \binom{m+1}{m-1} S_{m+2} - \cdots \pm \binom{N-1}{m-1} S_N \end{aligned} \quad (9)$$

where

$$S_m = \sum_{1 \le i_1 \le i_2 \cdots \le i_m \le N} P\left(A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_m}\right) . \quad (10)$$

P_m is the probability of at least m events occurring, and $P\left(A_{i_1} \cap A_{i_2} \cap \cdots \cap A_{i_m}\right)$ is the intersection probability of spike inputs $A_{i_1}, \ldots, A_{i_m}$ from the factor neurons. Note that P_m is different for each partition neuron since it depends on which factor neurons it receives spikes from. Spiking probability is given by P_1 and collision probability (the probability of at least two factors in a subset spiking simultaneously) is given by P_2. While this formula can be computed for small problems, it remains intractable for large m and N.

2) Partitioning Strategies: Given maximum indegree k_d, a factor base partitioning strategy specifies how to split the factor base into k_d subsets. Generally, the number of ways to partition a factor base with F elements into k_d subsets is the number of k_d-combinations, or $C(F-1, k_d)$. However, it may not make sense to consider the total number of factor base partitions. Instead, factors of similar magnitude should be kept in the same subset with reasoning as follows. The smoothness neuron sees only that a partition neuron is active; it doesn't know which factor or how many factors have spiked in each subset. This contributes to an information loss at the smoothness neuron. To minimize this loss amounts to minimizing the variance between factors in a particular subset, which leads to subsets that contain factors of similar magnitude.

Further justification for grouping factors of similar magnitude is based on the probability of spike collisions. The probability of any factor neuron being active is $\frac{2}{p^e}$ due to each factor having 2 polynomial roots (ignoring the effect of inhibition from higher powers). Mixing factors of different magnitude will cause spikes from smaller factors to frequently collide with spikes from the larger factors. In Figure 5, if 2 and 5^2 were both connected to the same partition neuron, that neuron would spike at least every other time step and it would not be known whether this was due to 2 or 5^2, which have very different spiking probabilities and contribute factors that differ by an order of magnitude to the sieving value.

Given that similar factors should be kept in the same subset, a partitioning strategy amounts to slicing an array of the sorted factors into k_d different slices using $k_d - 1$ indices, commonly referred to as the *multi-partitioning problem* [17]. To partition the factor base, the $k_d - 1$ indices are chosen so as to minimize the difference between the sums of a function $g(p^e)$ applied over each subset.

Four different partitioning strategies are evaluated which differ only in the way $g(p^e)$ is defined. The effect of each partition function is shown in Figure 6 using the same example

Algorithm 1 Partition an array G into k subsets of equal sum.

Input: sorted array G, number of subsets k
Output: list of $k-1$ indices idx

```
1.  idx = [], prev_idx = 0, remaining_sum = $\sum G$
2.  for i in range(k):
3.      target_sum = remaining_sum / (k - i)
4.      next_idx = prev_idx + 1
5.      acc = $G$[prev_idx]
6.      while abs(acc + $G$[next_idx] - target_sum)
7.                          < abs(acc - target_sum):
8.          acc += $G$[next_idx]
9.          next_idx += 1
10.     idx.append(next_idx)
11.     prev_idx = next_idx
12.     remaining_sum -= acc
13. return idx
```

from Section III under constraint $k_d = 3$.

Constant: The partition function is constant, $g(p^e) = 1$. This results in k_d subsets of equal size.

Logarithmic: Factors are partitioned by their logarithm such that $g(p^e) = \ln p^e$. This places smaller factors together and keeps larger factors apart, resulting in more spike collisions for smaller factors since these are both more frequent and co-occur in the same subsets.

Inverse logarithmic: Factors are partitioned by their inverse logarithm, $g(p^e) = \frac{1}{\ln p^e}$. This has the opposite effect of the logarithmic partition function, placing larger factors together and keeping smaller factors separate.

Probabilistic: Factors are partitioned according to their spiking probability, where $g(p^e) = \frac{2}{p^e}$ and $g(2) = 0.5$.

The partition problem is NP-complete. However, the restriction of keeping similar sized factors together leads to an efficient approximation which is a requirement in order to not increase the total runtime complexity of the sieving portion of integer factorization. Algorithm 1 greedily minimizes the difference between the sums of each subset in a single pass through the sorted array of factors. This is performed by stepping through the array and appending the next element to the current subset if the subset sum is closer to the target. When it is determined that adding the next element to the current subset is further from the target, a new subset is created and the target is updated by the remaining factors.

C. Limited Resolution and Topology

Given both shared synapse weights and maximum indegree, weight quantization and factor base partitioning must both be applied. This is only necessary if $F > k_d > k_w$. If $k_d \leq k_w$, then weight quantization is not necessary.

Weight quantization and factor base partitioning are performed in a two step process. First, the factor base is partitioned into k_d subsets using one of the partitioning strategies. The magnitude of each subset is approximated by the smallest factor which provides a set of reduced factors $\mathcal{F}'$ of size k_d. The weight quantization strategy is then applied to the reduced factors, treating them the same as the original factor base.

VI. Case Study

We modified `msieve` [18] to use the IBM Neurosynaptic System (NS1e) [15] for the sieving stage of integer factorization. `msieve` is a highly optimized publicly available implementation of the multiple polynomial quadratic sieve (MPQS, a variant of the quadratic sieve [19]) and NFS factoring algorithms. The core sieving procedure is optimized to minimize RAM access and arithmetic operations. Sieving intervals are broken up into blocks that fit into L1 processor cache using one byte per sieve value. The CPU host used in this work is a 2.6 GHz Intel Xeon E5-2670 which has an L1 cache size of 32 kB. Therefore, a sieving interval of length $M = 2^{17}$ would be broken up into four blocks that each fit into L1 cache.

The NS1e is a single-chip ASIC with 4096 cores and 256 LIF neurons per core, able to simulate a million spiking neurons while consuming under 100 mw at a normal operating frequency of 1 kHz. The TrueNorth architecture, implemented by the NS1e, specifies low-precision synaptic weights through shared axon types [20]. Each neuron can receive inputs from up to 256 axons, and each axon can be assigned one of four types. For each neuron, axons of the same type share a 9-bit signed weight. Thus, there can be at most 4 unique weights to any single neuron, and all 256 neurons on the same core must share the same permutation of axon types. These constraints require weight quantization for problems with greater than 4 factors ($k_w = 4$) and factor base partitioning for problems with greater than 256 factors ($k_d = 256$).

The neuron model in this work (Section II-B) is compatible with the TrueNorth architecture, which supports different modes of leak, reset behavior, and stochasticity [15]. Tonic spiking neurons are configured for prime powers up to 2^{18}, the maximum period that can be achieved by a single TrueNorth neuron with non-negative membrane potential. On a digital architecture, at least p^e distinct membrane potential values are needed to create a tonic spiking neuron with period p^e. On TrueNorth, the positive membrane potential threshold α is an unsigned 18-bit integer. The NS1e communicates with the CPU host via UDP, transmitting a single packet each time the smoothness neuron spikes. TrueNorth is a pipelined architecture, and the spikes from the tonic spiking neurons at time t are received by the smoothness neuron at time $t+2$ ($t+3$ with the partition layer). As a result, the smoothness neuron spikes when $f(t + x_{min} - 2)$ is likely smooth ($f(t + x_{min} - 3)$ with the partition layer).

A. Experimental Results

Results were obtained for n ranging from 32 to 128 bits. For each bit size, 100 cryptographically secure p and q of equal magnitude were generated using the Python Cryptography Toolkit [21] for a total of 9700 integer factorizations. For each n, B is set to $\exp\left(\frac{1}{2}\sqrt{\ln n \ln \ln n}\right)$. The factor base size b ranges from 8 for 32-bit n to 1224 for 128-bit n. Including prime powers, this requires 63 tonic spiking neurons for 32-bit n and 2707 tonic spiking neurons for 128-bit n, having 32 and 1354 connections to the smoothness neuron, respectively. The factor base size exceeds $k_d = 256$ at about 80 bits. Beyond this size, factor base partitioning must be employed using one of the strategies described in Section V-B. All the weight quantization and partitioning strategies were actually implemented and deployed to the NS1e except for

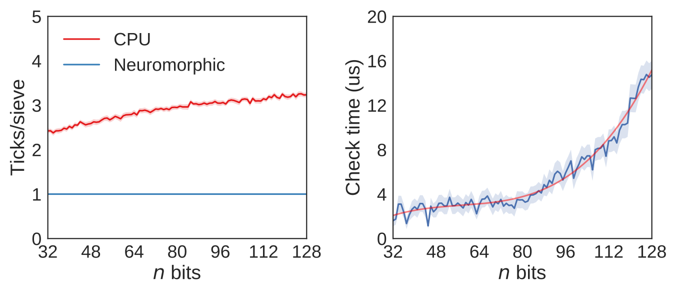


Fig. 7. Benchmark results showing CPU and NPU ticks per sieve value (left) and CPU check time per smooth candidate (right). Bands show 95% confidence interval (CI) obtained over 100 different n of equal magnitude.

the integer weight quantization which requires more than 4 unique weights.

msieve implements the MPQS, a variant of the quadratic sieve that utilizes multiple quadratic polynomials with unique coefficients [19]. This enables many smaller intervals to be sieved as opposed to a single large interval, leading to reduced overall magnitude of the polynomial values and a greater chance of containing a smooth number. For each polynomial, the sieving interval M is set to 2^{17}. Once the interval is exhausted, a new polynomial is generated and the sieving continues again with M set to 2^{17}. This process continues until at least $b + 1$ smooth numbers are detected for each n. The ground truth for smooth numbers is obtained by performing a definite check for smoothness for each polynomial value.

The neuromorphic sieving procedure is summarized as follows. For each polynomial for each n, the factor base primes are determined by msieve and then used to construct the neuromorphic sieve on the TrueNorth architecture, as described in Section III. Polynomial roots of the prime factors are calculated efficiently by the Tonelli-Shanks algorithm, and roots of prime powers are obtained through Hensel lifting. The resulting network is deployed to the NS1e, which runs for $M + 2$ time steps ($M + 3$ with partitioning). On each time step, if a smooth value is detected, a spike is generated and transmitted to the host which uses the timestamp to check the corresponding polynomial value for smoothness.

As a benchmark, the minimum number of CPU ticks per sieve value and the measured CPU check time are shown in Figure 7. The minimum CPU ticks per sieve value correspond to the minimum number of memory updates that need to be performed on the array that contains the sieving interval. Depending on the size of the sieving interval, additional ticks may be needed for memory access. Comparatively, the neuromorphic sieve spends exactly one tick per sieve value.

For each n, the check time per smooth candidate was measured with nanosecond resolution using differences between the clock_gettime function on a single dedicated core of the CPU host. These values are later used to determine the threshold of the smoothness neuron, which depends on the check time and NPU clock frequency; alternatively, the NPU clock frequency is determined by a desired spiking probability and check time (see Section IV-B). A polynomial of degree 3, determined to provide the best fit, is used to estimate the check times for larger problem sizes in the next section.

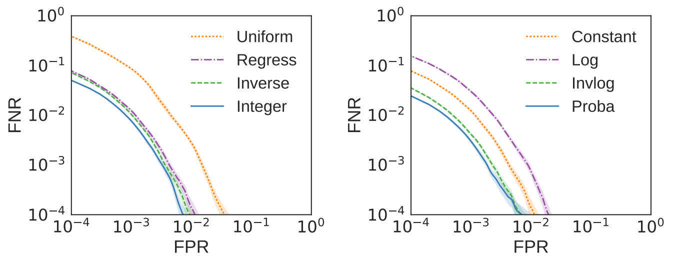


Fig. 8. ROC curves for 128-bit n comparing each weight quantization strategy (left) and partition function (right). Note that the CIs are relatively narrow.

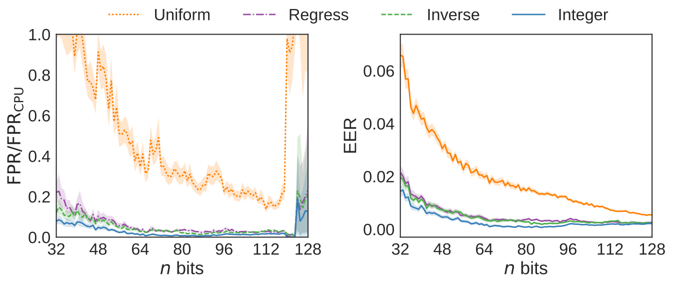


Fig. 9. Weight quantization strategy EER (right) and FPR compared to CPU FPR (left). Constant partitioning is used where necessary (above ~80 bits).

Figure 8 summarizes the results for 128-bit n, comparing the weight quantization and factor base partitioning strategies. At this size, both weight quantization and factor base partitioning are required. Each weight quantization strategy is evaluated using the constant partition function, and each factor base partitioning strategy uses regress weight quantization.

While the integer quantization strategy provides the best results with 0.21% EER (Figure 8, left), it is not compatible with TrueNorth which can accommodate at most 4 unique synapse weights. Despite this, performance of the regress and inverse quantization strategies approach that of the integer strategy, with 0.26% and 0.24% EERs, respectively. The uniform strategy performs the worst, having a 0.53% EER.

The factor base partitioning strategies for 128-bit n are compared in Figure 8 (right). The probabilistic partition function performs the best, with 0.14% EER, and the logarithmic partition function is the worst with 0.39% EER. Some possible explanations for this behavior are provided in the next section.

The weight quantization strategies are compared for increasing n bits in Figure 9 using constant partitioning where necessary (above ~80 bits). The FPR at the point on the ROC curve where the FNR equals the CPU FNR is determined and compared to the CPU FPR, given by $\mathrm{FPR}/\mathrm{FPR}_{\mathrm{CPU}}$ in Figure 9 (left). For most problem sizes, the regress, inverse, and integer strategies all have an FPR that is less than 20% of the CPU FPR, given equal FNRs. The FPR of the uniform strategy ranges from about 100% to 20% of the CPU FPR.

The EER of each quantization strategy is shown in Figure 9 (right). As n increases, the EER of each strategy generally decreases from 32 to 80 bits and slightly increases beyond 80 bits. This is expected since at this point the factor base size exceeds $k_d = 256$, the number of neurons that can be integrated in one step on TrueNorth, and factor base

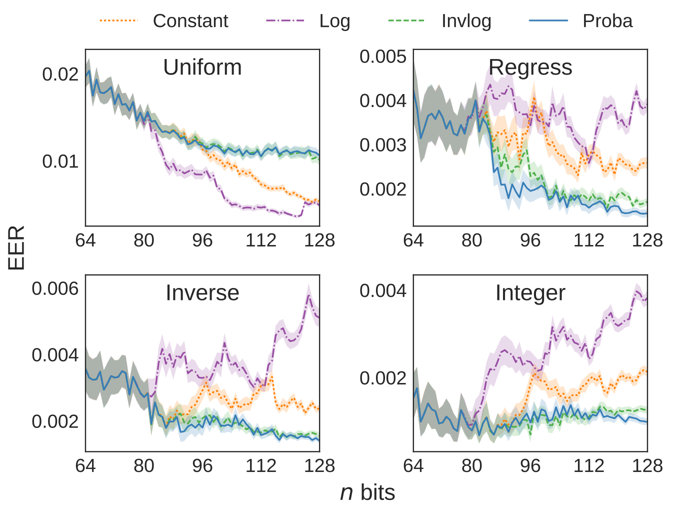


Fig. 10. Factor base partitioning strategy EER for each weight quantization strategy (given by plot titles). As n bits increases beyond the point at which factor base partitioning is required (~80 bits), the EER initially increases.

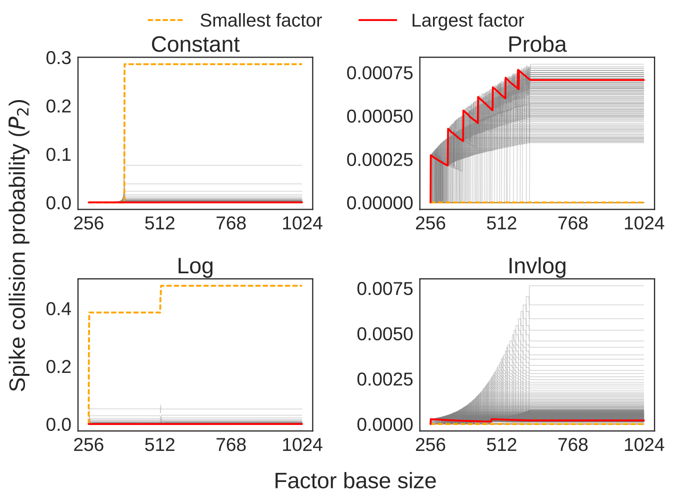


Fig. 11. Spike collision probabilities for each factor base partitioning strategy. Each line corresponds to one partition neuron. The partition neuron connected to the smallest factor is denoted by the dashed orange line and to the largest by the solid red line. Note the different y-axis scales.

partitioning must be employed. Despite this, good results are still obtained. A possible explanation for this behavior is provided in the next section in a simulation study aimed to better understand the asymptotic behavior of factor base partitioning.

The EER of each factor base partitioning strategy is shown in Figure 10 for increasing n under each weight quantization strategy. Results are only shown for n ranging from 64 to 128 bits since below this range factor base partitioning is not employed. Note that the partition strategy with the lowest EER depends on which weight quantization is used. With regress, inverse, and integer quantization, the inverse logarithmic and probabilistic partition functions have the best performance. With uniform weights, the opposite is true. In the best case weight scenario (integer weights), the EER of each partition strategy begins to increase beyond 80 bits, the point at which factor base partitioning is actually used. This is expected due to the occurrence of spike collisions. We verified that other ROC curve metrics, such as AUC, follow a similar trend.

VII. Simulation Study

A. Spike Collisions

In the previous section, it was shown that sieving performance (in terms of EER) did not significantly degrade as the problem size increased on the TrueNorth architecture. However, performance among the partition functions varied, with the EER of inverse logarithmic and probabilistic partitioning generally decreasing with increasing n bits and performing better than the constant and logarithmic strategies. To better understand why this is the case, we examined the spike collision probabilities of each strategy as the factor base size increases.

The theoretical probability of there being a spike collision in each of the 256 factor base subsets is given by P_2 (Equation 9). Recall from Section V-B that a collision occurs when two or more factors in a particular subset are active, sending two or more spikes to the respective partition neuron (Figure 5, third layer). Given the TrueNorth architecture with $k_d = 256$, P_2 is determined for each partition neuron (i.e., each subset) for increasing factor base size, shown in Figure 11.

In the constant and logarithmic partitioning strategies, the subset that contains the *smallest factor* has the *highest probability* of there being a spike collision, and the subset that contains the *largest factor* has the *lowest probability* of collision. As a result, information from the smaller (more frequent) factors is lost more often than the larger factors. Alternatively, in the inverse logarithmic and probabilistic strategies, the collision probability in the subset that contains the largest factor grows quicker than that of the smallest factor. However, the collision probabilities for most subsets in these strategies remain about an order of magnitude smaller than the constant and logarithmic strategies, leading to overall reduced spike collisions.

B. Scaling Up

A natural next step for neuromorphic sieving is to scale up to larger problem sizes by leveraging the sub-nanosecond response properties of near future neuromorphic devices; photonic architectures based on excitable graphene lasers operate on the order of picoseconds [22]. We estimated the sieving time of two theoretical neuromorphic architectures with 1 GHz and 1 THz clock rates, respectively. Both simulated architectures have 32-bit synaptic weights (using integer weight quantization) and have a maximum indegree of 256 (using constant factor base partitioning).

Table III shows the estimated sieving times on both theoretical architectures for several choices of n bits. For each problem size on each architecture, the threshold of the smoothness neuron is determined so as to maximize CPU utilization. This uses the lowest FNR that doesn't exceed 100% CPU utilization, where the check times for larger problem sizes are predicted by a best-fit polynomial of degree 3 to the

TABLE III
SIEVING TIMES ON THEORETICAL NEUROMORPHIC ARCHITECTURES.

n bits	F	t_c	P_s	1 GHz NPU	1 THz NPU
64	50	3 µs	0.0149	3 µs	4 ns
128	10^3	15 µs	0.0013	1 ms	1 us
256	10^5	246 µs	10^{-5}	4 sec	5 ms
512	10^8	3 ms	10^{-7}	18 days	30 mins
768	10^{10}	10 ms	10^{-9}	1500 years	2 years
1024	10^{13}	30 ms	10^{-11}	10^7 years	14,000 years

benchmark results in the previous section. The probability of a value being smooth is given by u^{-u}, where $u = \frac{\ln n}{2 \ln B}$, and the total sieving time is given by Equation 7 divided by the NPU clock frequency f, $T(n)/f$. F is estimated by $\frac{B}{2 \ln B}$, an approximation to half the number of primes below B.

Although factoring a 1024-bit integer may remain out of reach using a single NPU, sieving is easily parallelizable across multiple neuromorphic coprocessors as it requires very little communication from each processor to a central host. It is, therefore, not unreasonable to estimate a speedup proportional to the number of NPUs utilized. With tens of thousands of picosecond neuromorphic chips, a 1024-bit integer may be able to be factored in several years. For comparison, the 768-bit RSA took about 2000 CPU years to factor using the number field sieve (NFS), and the entire task was completed within 2 years using a cluster of several hundred machines [23].

VIII. DISCUSSION

While this work considered modern and near-future neuromorphic devices as a coprocessor for integer factorization, several special purpose devices for sieving have been proposed, and some of them have been implemented [24]. This includes a mix of analog and digital circuitry, all of which are application-specific architectures. In this section, we discuss some of these devices and how they relate to the neuromorphic sieve.

In 1999, Shamir proposed The Weizmann INstitute Key Locating Engine (TWINKLE), an optoelectronic sieving device with a flat topology [25]. Similar to the neuromorphic sieve, TWINKLE reverses the roles of space and time in the traditional CPU-based sieve. Primes in the factor base are represented by flashing light emitting diodes (LEDs), with one LED for each polynomial root. And like neurons in the neuromorphic sieve, the LEDs in TWINKLE each flash with period p and phase r for each root r of each prime p in the factor base. A single photodetector opposing the LEDs performs an analog summation in constant time by detecting the total luminosity emitted on each clock cycle. If the total luminosity exceeds a certain threshold, the sieving value for that clock cycle is smooth. Originally projected to have a clock rate of 10 GHz, and thus be able to sieve 10^8 values in 10 ms [25], it was later determined that a clock rate of 1 GHz is more realistic [26]. TWINKLE was estimated to cost several hundred thousand dollars to design and about $5000 to fabricate [25].

Shamir and Tromer later proposed The Weizmann Institute Relation Locator (TWIRL) in 2003, which, like TWINKLE, sieves over time instead of space [27]. Although both TWINKLE and TWIRL can be adapted for either the quadratic sieve or NFS, TWIRL was designed with the NFS in mind and has a few key differences from TWINKLE. Notably, TWIRL uses standard VLSI technology, aims to process many sieving locations in parallel, and partitions the factor base into three subsets (largish, small, and tiny) based on magnitude. Each subset is handled by a slightly different processing station. The output from the stations that handle small primes is compressed by a funnel that has a non-zero probability of overflow, analogous to a spike collision in the neuromorphic sieve when factor base partitioning is employed.

The neuromorphic sieve, like TWINKLE, aims to leverage a physics-based computation, specifically in performing a summation of the log factors. The ability to perform constant time summation is promised only by architectures that leverage the underlying physics of the device to compute, for example by integrating electrical potential (memristive architecture) or light intensity (photonic architecture). It is worth noting that while the NS1e conceptually provides a constant time synaptic integration, being a completely digital architecture, at the hardware level this is not the case.

Implementation on a continuous, or tickless, architecture remains a challenge. This work used a discrete neuron model which assumes that the architecture provides synchronization and noise-free neuron dynamics. We hypothesize that there are two main challenges in using a continuous neuron model, such as on a completely analog architecture with noise:

1) *Spike jitter*: In the continuous domain, it is not practical to exactly reproduce the time interval between two spikes. The period of a tonic spiking neuron may vary slightly from one moment to the next due to, e.g., noise and the inability to accurately measure time. In a digital clock, this type of noise is referred to as *clock jitter*, whereby the asymptotic frequency remains constant but the period between clock transition times varies slightly between cycles. The neuromorphic sieve requires exact integer periods and synchrony among the tonic neurons, thus time must be discretized and a time window around spikes established. The amount of noise will determine the size of the window and resolution with which multiple tonic neurons can be synchronized.

2) *Spike drift*: The frequency of a tonic neuron may drift over time. On a digital clock, this is referred to as *clock drift*. On a neuromorphic architecture, a similar effect could arise from the slightly differing time response properties between neurons, which are inherently asynchronous. As each tonic neuron experiences a different drift, the population will eventually go out of synchrony and the smoothness neuron loses the ability to account for all the factors on each step. There are, perhaps, some insights from clock synchronization that may be gained in addressing this issue.

These problems also raise several questions: how does sieving performance degrade due to spike jitter? Can reasonable performance still be obtained if neurons go out of synchronization? We leave these questions as future work in which the neuromorphic sieve is implemented on an analog architecture.

IX. Conclusions

This work highlights the ability of neuromorphic devices to perform computations other than machine learning. A $O(1)$ test for detecting smooth numbers was described, which, in most cases, is significantly more accurate than a CPU-based implementation that performs the same operation in $O(\ln \ln B)$ amortized time. Combined with the sub-nanosecond response properties of near-future neuromorphic chips, the factorization of integers approaching 1024-bits may become feasible within the next decade.

Besides speed, there are practical benefits to neuromorphic sieving. Current neuromorphic architectures, such as TrueNorth, offer cost-efficient computation, consuming orders of magnitude less energy than their CPU counterparts. Since integer factorization is dominated by the sieving effort, a neuromorphic coprocessor can be used to substantially decrease total energy consumption and cost.

There are several directions for future work. Currently, only the threshold of the smoothness neuron is varied, although this need not be the case. Partition neurons could have a threshold greater than 1, especially for subsets that contain many small factors causing them to spike if multiple factors in a subset spike. Future work should also aim to better understand the effect and asymptotic properties of factor base partitioning.

References

[1] C. Pomerance, "The role of smooth numbers in number theoretic algorithms," in *Proc. International Congress of Mathematicians (ICM)*, vol. 1, pp. 411–422, 1994.

[2] A. Granville, "Smooth numbers: computational number theory and beyond," *Algorithmic number theory: lattices, number fields, curves and cryptography*, vol. 44, pp. 267–323, 2008.

[3] J. V. Monaco and M. M. Vindiola, "Integer factorization with a neuromorphic sieve," in *Proc. 50th IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, 2017.

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[5] J. D. Dixon, "Asymptotically fast factorization of integers," *Mathematics of computation*, vol. 36, no. 153, pp. 255–260, 1981.

[6] C. Pomerance, "The quadratic sieve factoring algorithm," in *Proc. 1984 Workshop on the Theory and Application of Cryptographic Techniques (part of EUROCRYPT)*, pp. 169–182, Springer, 1984.

[7] P. L. Montgomery, "A block lanczos algorithm for finding dependencies over gf (2)," in *Proc. 1995 International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 106–120, Springer, 1995.

[8] R. Crandall and C. Pomerance, *Prime numbers: a computational perspective*, vol. 182. Springer, 2006.

[9] A. K. Lenstra, H. W. Lenstra Jr, M. S. Manasse, and J. M. Pollard, "The number field sieve," in *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 564–572, ACM, 1990.

[10] G. Indiveri and S.-C. Liu, "Memory and information processing in neuromorphic systems," *Proceedings of the IEEE*, vol. 103, no. 8, pp. 1379–1397, 2015.

[11] J. Backus, "Can programming be liberated from the von neumann style?: a functional style and its algebra of programs," *Communications of the ACM*, vol. 21, no. 8, pp. 613–641, 1978.

[12] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[13] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[14] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proc. 2015 Advances in Neural Information Processing Systems (NIPS)*, pp. 3123–3131, 2015.

[15] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, *et al.*, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[16] W. Feller, *An Introduction to Probability Theory and its Applications, Vol. 1.* New York, NY: Wiley, second ed., 1959.

[17] R. E. Korf, "Multi-way number partitioning," in *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[18] J. Papadopoulos, "msieve." https://sourceforge.net/projects/msieve/. Version 1.52.

[19] R. D. Silverman, "The multiple polynomial quadratic sieve," *Mathematics of Computation*, vol. 48, no. 177, pp. 329–339, 1987.

[20] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[21] D. Litzenberger, "Python cryptography toolkit." https://www.dlitz.net/software/pycrypto/. Version 2.6.1.

[22] M. A. Nahmias, B. J. Shastri, A. N. Tait, and P. R. Prucnal, "A leaky integrate-and-fire laser neuron for ultrafast cognitive computing," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 19, no. 5, pp. 1–12, 2013.

[23] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik, *et al.*, "Factorization of a 768-bit rsa modulus," in *Proc. 30th International Cryptology Conference (CRYPTO)*, pp. 333–350, Springer, 2010.

[24] SHARCS 2012, "Workshop on special-purpose hardware for attacking cryptographic systems." http://www.sharcs.org/, 2012.

[25] A. Shamir, "Factoring large numbers with the twinkle device," in *Proc. International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, pp. 2–12, Springer, 1999.

[26] A. K. Lenstra and A. Shamir, "Analysis and optimization of the twinkle factoring device," in *Proc. 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pp. 35–52, Springer, 2000.

[27] A. Shamir and E. Tromer, "Factoring large numbers with the twirl device," in *Proc. 23rd Annual International Cryptology Conference (CRYPTO)*, pp. 1–26, Springer, 2003.

John V. Monaco is a Computer Scientist at the Computational and Information Sciences Directorate of the U.S. Army Research Laboratory and Adjunct Professor at Pace University. Dr. Monaco earned his BS in Computer Science and Mathematics (2012), MS in Computer Science (2013), and PhD in Computer Science (2015) from Pace University. In 2011, he was selected for the highly competitive and prestigious Information Assurance Scholarship Program by the U.S. Department of Defense. In 2013, Dr. Monaco was named one of Westchester's "Top Professionals under 30" for research in keystroke biometrics at Pace University. He has since achieved first place in several international biometric competitions. His primary areas of interest are behavioral biometrics and neuromorphic computing.

Manuel M. Vindiola is a Cognitive Scientist at the Computational and Information Sciences Directorate of the U.S. Army Research Laboratory. Dr.Vindiola received his BS in Cognitive Science and Linguistics (2003) at the University of California, Berkeley, and his MS in Cognitive Science (2005) and PhD in Cognitive Science (2013) at Johns Hopkins University.