Appears in the Proceedings of the 41st International Symposium on Computer Architecture, 2014

General-Purpose Code Acceleration with Limited-Precision Analog Computation

Renée St. Amant[⋆] Amir Yazdanbakhsh[§] Jongse Park[§] Bradley Thwaites[§]
Hadi Esmaeilzadeh[§] Arjang Hassibi[⋆] Luis Ceze[†] Doug Burger[‡]

[⋆]University of Texas at Austin [§]Georgia Institute of Technology
[†]University of Washington [‡]Microsoft Research

stamant@cs.utexas.edu a.yazdanbakhsh@gatech.edu jspark@gatech.edu bthwaites@gatech.edu
hadi@cc.gatech.edu arjang@mail.utexas.edu luisceze@cs.washington.edu dburger@microsoft.com

Abstract

As improvements in per-transistor speed and energy efficiency diminish, radical departures from conventional approaches are becoming critical to improving the performance and energy efficiency of general-purpose processors. We propose a solution—from circuit to compiler—that enables general-purpose use of limited-precision, analog hardware to accelerate "approximable" code—code that can tolerate imprecise execution. We utilize an algorithmic transformation that automatically converts approximable regions of code from a von Neumann model to an "analog" neural model. We outline the challenges of taking an analog approach, including restricted-range value encoding, limited precision in computation, circuit inaccuracies, noise, and constraints on supported topologies. We address these limitations with a combination of circuit techniques, a hardware/software interface, neural-network training techniques, and compiler support. Analog neural acceleration provides whole application speedup of 3.7× and energy savings of 6.3× with quality loss less than 10% for all except one benchmark. These results show that using limited-precision analog circuits for code acceleration, through a neural approach, is both feasible and beneficial over a range of approximation-tolerant, emerging applications including financial analysis, signal processing, robotics, 3D gaming, compression, and image processing.

1. Introduction

Energy efficiency now fundamentally limits microprocessor performance gains. CMOS scaling no longer provides gains in efficiency commensurate with transistor density increases [16, 25]. As a result, both the semiconductor industry and the research community are increasingly focused on specialized accelerators, which provide large gains in efficiency and performance by restricting the workloads that benefit. The community is facing an "iron triangle"; we can choose any two of performance, efficiency, and generality at the expense of the third. Before the effective end of Dennard scaling, we improved all three consistently for decades. Solutions that improve performance and efficiency, while retaining as much generality as possible, are highly desirable, hence the growing interest in GPGPUs and FPGAs. A growing body of recent work [13, 17, 44, 2, 35, 8, 28, 38, 18, 51, 46] has focused on *approximation* as a strategy for the iron triangle. Many classes of applications can tolerate small errors in their outputs with no discernible loss in *QoR* (Quality of Result). Many conventional techniques in energy-efficient computing navigate a design space defined by the two dimensions of performance and energy, and traditionally trade one for the other. General-purpose approximate computing explores a third dimension—that of error.

Many design alternatives become possible once precision is relaxed. An obvious candidate is the use of analog circuits for computation. However, computation in the analog domain has several major challenges, even when small errors are permissible. First, analog circuits tend to be special purpose, good for only specific operations. Second, the bit widths they can accommodate are smaller than current floating-point standards (i.e. 32/64 bits), since the ranges must be represented by physical voltage or current levels. Another consideration is determining where the boundaries between digital and analog computation lie. Using individual analog operations will not be effective due to the overhead of A/D and D/A conversions. Finally, effective storage of temporary analog results is challenging in current CMOS technologies. These limitations has made it ineffective to design analog von Neumann processors that can be programmed with conventional languages.

Despite these challenges, the potential performance and energy gains from analog execution are highly attractive. An important challenge is thus to architect designs where a significant portion of the computation can be run in the analog domain, while also addressing the issues of value range, domain conversions, and relative error. Recent work on Neural Processing Units (NPUs) may provide a possible approach [18]. NPU-enabled systems rely on an algorithmic transformation that converts regions of approximable general-purpose code into a neural representation (specifically, multi-layer perceptrons) at compile time. At run-time, the processor invokes the NPU instead of running the original code. NPUs have shown large performance and efficiency gains, since they subsume an entire code region (including all of the instruction fetch, decode, etc., overheads). They have an added advantage in that they convert many distinct code patterns into a common representation that can be run on a single physical accelerator, improving generality.

NPUs may be a good match for mixed-signal implementations for a number of reasons. First, prior research has shown that neural networks can be implemented in analog domain to solve classes of domain-specific problems, such as pattern recognition [5, 47, 49, 32]. Second, the process of invoking a neural network and returning a result defines a clean, coarse-grained interface for D/A and A/D conversion. Third, the compile-time training of the network permits any analog-specific restrictions to be hidden from the programmer. The programmer simply specifies which region of the code can be approximated, without adding any neural-network-specific information. Thus, no additional changes to the programming model are necessary.

In this paper we evaluate an NPU design with mixed-signal components and develop a compilation workflow for utilizing the mixed-signal NPU for code acceleration. The goal of this study is to investigate challenges and define potential solutions to enable effective mixed-signal NPU execution. The objective is to both bound application error to sufficiently low levels and achieve worthwhile performance or efficiency gains for general-purpose approximable code. This study makes the following four findings:

1. Due to range limitations, it is necessary to limit the scope of the analog execution to a single neuron; inter-neuron communication should be in the digital domain.
2. Again due to range issues, there is an interplay between the bit widths (inputs and weights) that neurons can use and the number of inputs that they can process. We found that the best design limited weights and inputs to eight bits, while also restricting the number of inputs to each neuron to eight. The input count limitation restricts the topological space of feasible neural networks.
3. We found that using a customized continuous-discrete learning method (CDLM) [10], which accounts for limited-precision computation at training time, is necessary to reduce error due to analog range limitations.
4. Given the analog-imposed topology restrictions, we found that using a Resilient Back Propagation (RPROP) [30] training algorithm can further reduce error over a conventional backpropagation algorithm.

We found that exposing the analog limitations to the compiler allowed for the compensation of these shortcomings and produced sufficiently accurate results. The latter three findings were all used at training time; we trained networks at compile time using 8-bit values, topologies restricted to eight inputs per neuron, plus RPROP and CDLM for training. Using these techniques together, we were able to bound error on all applications but one to a 10% limit, which is commensurate with entirely digital approximation techniques. The average time required to compute a neural result was 3.3× better than a previous digital implementation with an additional energy savings of 12.1×. The performance gains result in an average full-application-level improvement of 3.7× and 23.3× in performance and energy-delay product, respectively. This study shows that using limited-precision analog circuits for code acceleration, by converting regions of imperative code to neural networks and exposing the circuit limitations to the compiler, is both feasible and advantageous. While it may be possible to move more of the accelerator architecture design into the analog domain, the current mixed-signal design performs well enough that only 3% and 46% additional improvements in application-level energy consumption and performance are possible with improved accelerator designs. However, improving the performance of the analog NPU may lead to higher overall performance gains.

2. Overview and Background

Programming. We use a similar programming model as described in [18] to enable programmers to mark error-tolerant regions of code as candidates for transformation using a simple keyword, `approximable`. Explicit annotation of code for approximation is a common practice in approximate programming languages [45, 7]. A candidate region is an error-tolerant function of any size, containing function calls, loops, and complex control flow. Frequently executed functions provide a greater opportunity for gains. In addition to error tolerance, the candidate function must have well-defined inputs and outputs. That is, the number of inputs and outputs must be known at compile time. Additionally, the code region must not read any data other than its inputs, nor affect any data other than its outputs. No major changes are necessary to the programming language beyond adding the approximable keyword.

Exposing analog circuits to the compiler. Although an analog accelerator presents the opportunity for gains in efficiency over a digital NPU, it suffers from reduced accuracy and flexibility, which results in limitations on possible network topologies and limited-precision computation, potentially resulting in a decreased range of applications that can utilize the acceleration. These shortcomings at the hardware level, however, can be exposed as a high-level model and considered in the training phase.

Four characteristics need to be exposed: (1) limited precision for input and output encoding, (2) limited precision for encoding weights, (3) the behavior of the activation function (sigmoid), (4) limited feasible neural topologies. Other low-level circuit behavior such as response to noise can also be exposed to the compiler. Section 5 describes this necessary hardware/software interface in more detail.

Analog neural accelerator circuit design. To extract the high-level model for the compiler and to be able to accelerate execution, we design a mixed-signal neural hardware for multilayer perceptrons. The accelerator must support a large enough variety of neural network topologies to be useful over a wide range of applications. As we will show, each applications requires a different topology for the neural network that is replacing its approximable regions of code. Section 4 describes a candidate A-NPU circuit design, and outlines the challenges and tradeoffs present with an analog implementation.

Compiling for analog neural hardware. The compiler aims to mimic approximable regions of code with neural networks that can be executable on the mixed-signal accelerator. While considering the limitation of the analog hardware, the compiler searches the topology space of the neural networks and selects and trains a neural network to produce outputs

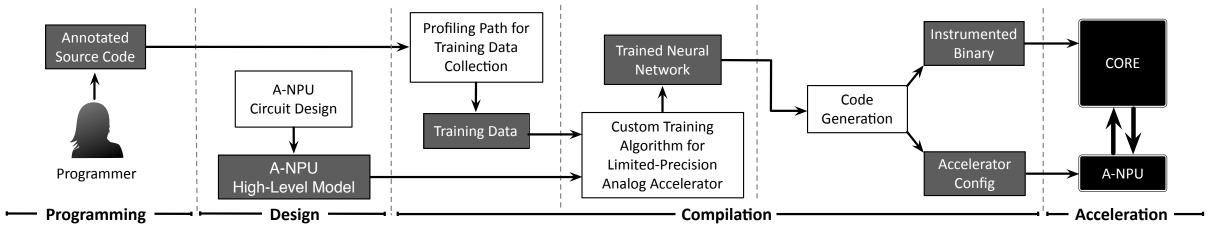


Figure 1: Framework for using limited-precision analog computation to accelerate code written in conventional languages.

comparable to those produced by the original code segment.

1) Profile-driven training data collection. During a profiling stage, the compiler runs the application with representative profiling inputs and collects the inputs and outputs to the approximable code region. This step provides the training data for the rest of the compilation workflow.

2) Training for a limited-precision A-NPU. This stage is where our compilation workflow significantly deviates from the framework presented in [18] that targets digital NPUs. The compiler uses the collected training data to train a multilayer perceptron neural network, choosing a network topology, i.e. the number of neurons and their connectivity, and taking a gradient descent approach to find the synaptic weights of the network while minimizing the error with respect to the training data. This compilation stage does a neural topology search to find the smallest neural network that (a) adheres to the organization of the analog circuit and (b) delivers acceptable accuracy at the application level. The network training algorithm, which finds optimal synaptic weights, uses a combination of a resilient back propagation algorithm, RPROP [30], that we found to outperform traditional back propagation for restricted network topologies, and a continuous-discrete learning method, CDLM [10], that attempts to correct for error due to limited-precision computation. Section 5 describes these techniques that address analog limitations.

3) Code generation for hybrid analog-digital execution. Similar to prior work [18], in the code generation phase, the compiler replaces each instance of the original program code with code that initiates a computation on the analog neural accelerator. Similar ISA extensions are used to specify the neural network topology, send input and weight values to the A-NPU, and retrieve computed outputs from the A-NPU.

3. Analog Circuits for Neural Computation

This section describes how analog circuits can perform the computation of neurons in multi-layer perceptrons, which are widely used neural networks. We also discuss, at a high-level, how limitations of the analog circuits manifest in the computation. We explain how these restrictions are exposed to the compilation framework. The next section presents a concrete design for the analog neural accelerator.

As Figure 2a illustrates, each neuron in a multi-layer perceptron takes in a set of inputs (x_i) and performs a weighted sum of those input values ($\sum_i x_i w_i$). The weights (w_i) are the result of training the neural network on . After the summation

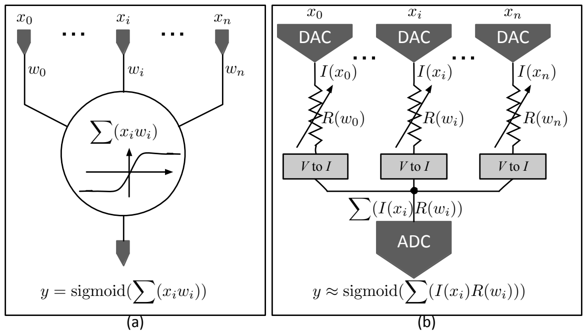


Figure 2: One neuron and its conceptual analog circuit.

stage, which produces a linear combination of the weighted inputs, the neuron applies a nonlinearity function, *sigmoid*, to the result of summation.

Figure 2b depicts a conceptual analog circuit that performs the three necessary operations of a neuron: (1) scaling inputs by weight ($x_i w_i$), (2) summing the scaled inputs ($\sum_i x_i w_i$), and (3) applying the nonlinearity function (*sigmoid*). This conceptual design first encodes the digital inputs (x_i) as analog current levels ($I(x_i)$). Then, these current levels pass through a set of variable resistances whose values ($R(w_i)$) are set proportional to the corresponding weights (w_i). The voltage level at the output of each resistance ($I(x_i)R(w_i)$), is proportional to $x_i w_i$. These voltages are then converted to currents that can be summed quickly according to Kirchhoff's current law (KCL). Analog circuits only operate linearly within a small range of voltage and current levels, outside of which the transistors enter saturation mode with IV characteristics similar in shape to a non-linear sigmoid function. Thus, at the high level, the non-linearity is naturally applied to the result of summation when the final voltage reaches the analog-to-digital converter (ADC). Compared to a digital implementation of a neuron, which requires multipliers, adder trees and sigmoid lookup tables, the analog implementation leverages the physical properties of the circuit elements and is orders of magnitude more efficient. However, it operates in limited ranges and therefore offers limited precision.

Analog-digital boundaries. The conceptual design in Figure 2b draws the analog-digital boundary at the level of an algorithmic neuron. As we will discuss, the analog neural accelerator will be a composition of these analog neural units (ANUs). However, an alternative design, primarily optimizing for efficiency, may lay out the entirety of a neural network with only analog components, limiting the D-to-A and A-to-D conversions to the inputs and outputs of the neural network and not the individual neurons. The overhead of conversions in the ANUs significantly limits the potential efficiency gains of an analog approach toward neural computation. However, there is a tradeoff between efficiency, reconfigurability (generality), and accuracy in analog neural hardware design. Pushing more of the implementation into the analog domain gains efficiency at the expense of flexibility, limiting the scope of supported network topologies and, consequently, limiting potential network accuracy. The NPU approach targets *code approximation*, rather than typical, simpler neural tasks, such as recognition and prediction, and imposes higher accuracy requirements. The main challenge is to manage this tradeoff to achieve acceptable accuracy for code acceleration, while delivering higher performance and efficiency when analog neural circuits are used for *general-purpose code acceleration*.

As prior work [18] has shown and we corroborate, regions of code from different applications require different topologies of neural networks. While a holistically analog neural hardware design with fixed-wire connections between neurons may be efficient, it effectively provides a fixed topology network, limiting the scope of applications that can benefit from the neural accelerator, as the optimal network topology varies with application. Additionally, routing analog signals among neurons and the limited capability of analog circuits for buffering signals negatively impacts accuracy and makes the circuit susceptible to noise. In order to provide additional flexibility, we set the digital-analog boundary in conjunction with an algorithmic, sigmoid-activated neuron. where a set of digital inputs and weights are converted to the analog domain for efficient computation, producing a digital output that can be accurately routed to multiple consumers. We refer to this basic computation unit as an analog neural unit, or ANU. ANUs can be composed, in various physical configurations, along with digital control and storage, to form a reconfigurable mixed-signal NPU, or A-NPU.

One of the most prevalent limitations in analog design is the *bounded range* of currents and voltages within which the circuits can operate effectively. These range limitations restrict the bit-width of input and weight values and the network topologies that can be computed accurately and efficiently. We expose these limitations to the compiler and our custom training algorithm and compilation workflow considers these restrictions when searching for optimal network topologies and training neural networks. As we will show, one of the insights from this work is that even with limited bit-width (≤ 8), and a restricted neural topology, many general-purpose approximate applications achieve acceptable accuracy and significantly benefit from mixed-signal neural acceleration.

Value representation and bit-width limitations. One of the fundamental design choices for an ANU is the bit-width of inputs and weights. Increasing the number of bits results in an exponential increase in the ADC and DAC energy dissipation and can significantly limit the benefits from analog acceleration. Furthermore, due to the fixed range of voltage and current levels, increasing the number of bits translates to quantizing this fixed value range to fine granularities that practical ADCs can not handle. In addition, the fine granularity encoding makes the analog circuit significantly more susceptible to noise, thermal, voltage, current, and process variations. In practice, these non-ideal effects can adversely affect the final accuracy when more bit-width is used for weights and inputs. We design our ANUs such that the granularity of the voltage and current levels used for information encoding is to a large degree robust to variations and noise.

Topology restrictions. Another important design choice is the *number of inputs* in the ANU. Similar to bit-width, increasing the number of ANU inputs translates to encoding a larger value range in a bounded voltage and current range, which, as discussed, becomes impractical. There is a tradeoff between accuracy and efficiency in choosing the number ANU inputs. The larger the number of inputs, the larger the number of multiply and add operations that can be done in parallel in the analog domain, increasing efficiency. However, due to the bounded range of voltage and currents, increasing the number of inputs requires decreasing the number of bits for inputs and weights. Through circuit-level simulations, we empirically found that limiting the number of inputs to eight with 8-bit inputs and weights strikes a balance between accuracy and efficiency. A digital implementation does not impose such restrictions on the number of inputs to the hardware neuron and it can potentially compute arbitrary topologies of neural networks. However, this unique ANU limitation restricts the topology of the neural network that can run on the analog accelerator. Our customized training algorithm and compilation workflow takes into account this topology limitation and produces neural networks that can be computed on our mixed-signal accelerator.

Non-ideal sigmoid. The saturation behavior of the analog circuit that leads to sigmoid-like behavior after the summation stage represents an approximation of the ideal sigmoid. We measure this behavior at the circuit level and expose it to the compiler and the training algorithm.

4. Mixed-Signal Neural Accelerator (A-NPU)

This section describes a concrete ANU design and the mixed-signal, neural accelerator, A-NPU.

4.1. ANU Circuit Design

Figure 3 illustrates the design of a single analog neuron (ANU). The ANU performs the computation of one neuron, or $y \approx sigmoid(\sum_i w_i x_i)$. We place the analog-digital boundary at the ANU level, with computation in the analog domain and storage in the digital domain. Digital input and weight values are represented in sign-magnitude form. In the figure, s_{w_i} and s_{x_i} represent the sign bits and w_i and x_i represent the magnitude. Digital input values are converted to the analog domain through current-steering DACs that translate digital values to analog currents. Current-steering DACs are used for

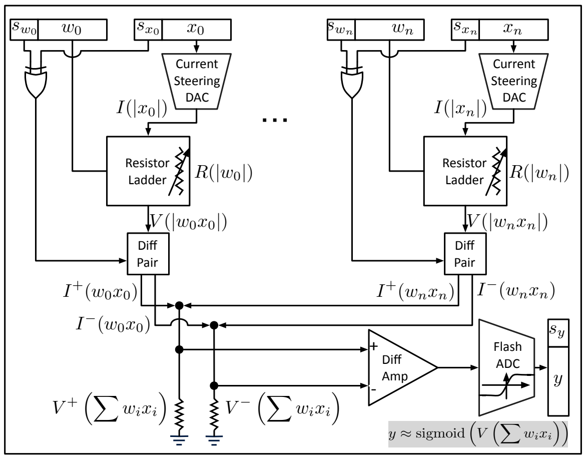


Figure 3: A single analog neuron (ANU).

their speed and simplicity. In Figure 3, $I(|x_i|)$ is the analog current that represents the magnitude of the input value, x_i. Digital weight values control resistor-string ladders that create a variable resistance depending on the magnitude of each weight ($R(|w_i|)$) . We use a standard resistor ladder thats consists of a set of resistors connected to a tree-structured set of switches. The digital weight bits control the switches, adjusting the effective resistance, $R(|w_i|)$, seen by the input current ($I(|x_i|)$). These variable resistances scale the input currents by the digital weight values, effectively multiplying each input magnitude by its corresponding weight magnitude. The output of the resistor ladder is a voltage: $V(|w_ix_i|) = I(|x_i|) \times R(|w_i|)$. The resistor network requires 2^m resistors and approximately 2^{m+1} switches, where m is the number of digital weight bits. This resistor ladder design has been shown to work well for $m \leq 10$. Our circuit simulations show that only minimally sized switches are necessary.

$V(|w_ix_i|)$, as well as the XOR of the weight and input sign bits, feed to a differential pair that converts voltage values to two differential currents ($I^+(w_ix_i)$, $I^-(w_ix_i)$) that capture the sign of the weighted input. These differential currents are proportional to the voltage applied to the differential pair, $V(|w_ix_i|)$. If the voltage difference between the two gates is kept small, the current-voltage relationship is linear, producing $I^+(w_ix_i) = \frac{I_{bias}}{2} + \Delta I$ and $I^-(w_ix_i) = \frac{I_{bias}}{2} - \Delta I$. Resistor ladder values are chosen such that the gate voltage remains in the range that produces linear outputs, and consequently a more accurate final result. Based on the sign of the computation, a switch steers either the current associated with a positive value or the current associated with a negative value to a single wire to be efficiently summed according to Kirchhoff's current law. The alternate current is steered to a second wire, retaining differential operation at later design stages. Differential operation combats environmental noise and increases gain, the later being particularly important for mitigating the impact of analog range challenges at later stages.

Resistors convert the resulting pair of differential currents to voltages, $V^+(\sum_i w_ix_i)$ and $V^-(\sum_i w_ix_i)$, that represent the weighted sum of the inputs to the ANU. These voltages are

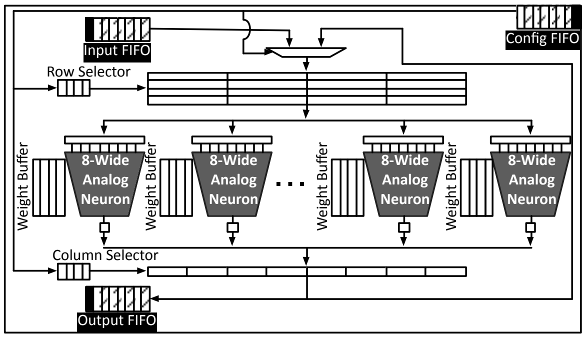


Figure 4: Mixed-signal neural accelerator, A-NPU. Only four of the ANUs are shown. Each ANU processes eight 8-bit inputs.

used as input to an additional amplification stage (implemented as a current-mode differential amplifier with diode-connected load). The goal of this amplification stage is to significantly magnify the input voltage *range of interest* that maps to the linear output region of the desired sigmoid function. Our experiments show that neural networks are sensitive to the steepness of this non-linear function, losing accuracy with shallower, non-linear activation functions. This fact is relevant for an analog implementation because steeper functions increase range pressure in the analog domain, as a small range of interest must be mapped to a much larger output range in accordance with ADC input range requirements for accurate conversion. We magnify this range of interest, choosing circuit parameters that give the required gain, but also allowing for saturation with inputs outside of this range.

The amplified voltage is used as input to an ADC that converts the analog voltage to a digital value. We chose a flash ADC design (named for its speed), which consists of a set of reference voltages and comparators [1, 31]. The ADC requires 2^n comparators, where n is the number of digital output bits. Flash ADC designs are capable of converting 8 bits at a frequency on the order of one GHz. We require 2–3 mV between ADC quantization levels for accurate operation and noise tolerance. Typically, ADC reference voltages increase linearly; however, we use a non-linearly increasing set of reference voltages to capture the behavior of a sigmoid function, which also improves the accuracy of the analog sigmoid.

4.2. Reconfigurable Mixed-Signal A-NPU

We design a reconfigurable mixed-signal A-NPU that can perform the computation of a wide variety of neural topologies since each requires a different topology. Figure 4 illustrates the A-NPU design with some details omitted for clarity. The figure shows four ANUs while the actual design has eight. The A-NPU is a time-multiplexed architecture where the algorithmic neurons are mapped to the ANUs based on a static scheduling algorithm, which is loaded to the A-NPU before invocation. The multi-layer perceptron consists of layers of neurons, where the inputs of each layer are the outputs of the previous layer. The ANU starts from the input layer and performs the computations of the neurons layer by layer. The Input Buffer always contains the inputs to the neurons, either coming from the processor or from the previous layer computation. The Output Buffer, which is a single entry buffer, collects the outputs of the ANUs. When all of its columns are computed, the results are pushed back to the Input Buffer to enable calculation of the next layer. The Row Selector determines which entry of the input buffer will be fed to the ANUs. The output of the ANUs will be written to a single-entry output buffer. The Column Selector determines which column of the output buffer will be written by the ANUs. These selectors are FIFO buffers whose values are part of the preloaded A-NPU configuration. All the buffers are digital SRAM structures.

Each ANU has eight inputs. As shown in Figure 4, each A-NPU is augmented with a dedicated weight buffer, storing the 8-bit weights. The weight buffers simultaneously feed the weights to the ANUs. The weights and the order in which they are fed to the ANUs are part of the A-NPU configuration. The Input Buffer and Weight Buffers synchronously provide the inputs and weights for the ANUs based on the pre-loaded order.

A-NPU configuration. During code generation, the compiler produces an A-NPU configuration that constitutes the weights and the schedule. The static A-NPU scheduling algorithm first assigns an order to the neurons of the neural network, in which the neurons will be computed in the ANUs. The scheduler then takes the following steps for each layer of the neural network: (1) Assign each neuron to one of the ANUs. (2) Assign an order to neurons. (3) Assign an order to the weights. (4) Generate the order for inputs to feed the ANUs. (5) Generate the order in which the outputs will be written to the Output Buffer. The scheduler also assigns a unique order for the inputs and outputs of the neural network in which the core communicates data with the A-NPU.

4.3. Architectural interface for A-NPU

We adopt the same FIFO-based architectural interface through which a digital NPU communicates with the processor [18]. The A-NPU is tightly integrated to the pipeline. The processor only communicates with the ANUs through the Input, Output, Config FIFOs. The processor ISA is extended with special instructions that can enqueue and dequeue data from these FIFOs as shown in Figure 4. When a data value is queued/dequeued to/from the Input/Output FIFO, the A-NPU converts the values to the appropriate representation for the A-NPU/processor.

5. Compilation for Analog Acceleration

As Figure 1 illustrates, the compilation for A-NPU execution consists of three stages: (1) profile-driven data collection, (2) training for a limited-precision A-NPU, and (3) code generation for hybrid analog-digital execution. In the profile-driven data collection stage, the compiler instruments the application to collect the inputs and outputs of approximable functions. The compiler then runs the application with representative inputs and collects the inputs and their corresponding outputs. These input-output pairs constitute the training data. Section 4 briefly discussed ISA extensions and code generation. While compilation stages (1) and (3) are similar to the techniques presented for a digital implementation [18], the training phase is unique to an analog approach, accounting for analog-imposed, topology restrictions and adjusting weight selection to account for limited-precision computation.

Hardware/software interface for exposing analog circuits to the compiler. As we discussed in Section 3, we expose the following analog circuit restrictions to the compiler through a hardware/software interface that captures the following circuit characteristics: (1) input bit-width limitations, (2) weight bit-width limitations, (3) limited number of inputs to each analog neuron (topology restriction), and (4) the non-ideal shape of the analog sigmoid. The compiler internally constructs a high-level model of the circuit based on these limitations and uses this model during the neural topology search and training with the goal of limiting the impact of inaccuracies due to an analog implementation.

Training for limited bit widths and analog computation. Traditional training algorithms for multi-layered perceptron neural networks use a gradient descent approach to minimize the average network error, over a set of training input-output pairs, by backpropagating the output error through the network and iteratively adjusting the weight values to minimize that error. Traditional training techniques, however, that do not consider limited-precision inputs, weights, and outputs perform poorly when these values are saturated to adhere to the bit-width requirements that are feasible for an implementation in the analog domain. Simply limiting weight values during training is also detrimental to achieving quality outputs because the algorithm does not have sufficient precision to converge to a quality solution.

To incorporate bit-width limitations into the training algorithm, we use a customized continuous-discrete learning method (CDLM) [10]. This approach takes advantage of the availability of full-precision computation at training time and then adjusts slightly to optimize the network for errors due to limited-precision values. In an initial phase, CDLM first trains a fully-precise network according to a standard training algorithm, such as backpropagation [43]. In a second phase, it discretizes the input, weight, and output values according the the exposed analog specification. The algorithm calculates the new error and backpropagates that error through the fully-precise network using full-precision computation and updates the weight values according to the algorithm also used in stage 1. This process repeats, backpropagating the 'discrete' errors through a precise network. The original CDLM training algorithm was developed to mitigate the impact of limited-precision weights. We customize this algorithm by incorporating the input bit-width limitation and the output bit-width limitation in addition to limited weight values. Additionally, this training scheme is advantageous for an analog implementation because it is general enough to also make up for errors that arise due to an analog implementation, such as a non-ideal sigmoid function and any other analog non-ideality that behaves consistently.

In essence, after one round of full-precision training, the compiler models an analog-like version of the network. A second, CDLM-based training pass adjusts for these analog-imposed errors, enabling the inaccurate and limited A-NPU as an option for a beneficial NPU implementation by maintaining acceptable accuracy and generality.

Training with topology restrictions. In addition to determining weight values for a given network topology, the compiler searches the space of possible topologies to find an optimal network for a given approximable code region. Conventional multi-layered perceptron networks are fully connected, i.e. the output of each neuron in one layer is routed to the input of each neuron in the following layer. However, analog range limitations restrict the number of inputs that can be computed in a neuron (eight in our design). Consequently, network connections must be limited, and in many cases, the network can not be fully connected.

We impose the circuit restriction on the connectivity between the neurons during the topology search and we use a simple algorithm guided by the mean-squared error of the network to determine the best topology given the exposed restriction. The error evaluation uses a typical cross-validation approach: the compiler partitions the data collected during profiling into a *training set*, 70% of the data, and a *test set*, the remaining 30%. The topology search algorithm trains many different neural-network topologies using the training set and chooses the one with the highest accuracy on the test set and the lowest latency on the A-NPU hardware (prioritizing accuracy). The space of possible topologies is large, so we restrict the search to neural networks with at most two hidden layers. We also limit the number of neurons per hidden layer to powers of two up to 32. The numbers of neurons in the input and output layers are predetermined based on the number of inputs and outputs in the candidate function.

To further improve accuracy, and compensate for topology-restricted networks, we utilize a Resilient Back Propagation (RPROP) [30] training algorithm as the base training algorithm in our CDLM framework. During training, instead of updating the weight values based on the backpropagated error (as in conventional backpropagation [43]), the RPROP algorithm increases or decreases the weight values by a predefined value based on the sign of the error. Our investigation showed that RPROP significantly outperforms conventional backpropagation for the selected network topologies, requiring only half of the number of training epochs as backpropagation to converge on a quality solution. The main advantage of the application of RPROP training to an analog approach to neural computing is its robustness to the sigmoid function and topology restrictions imposed by the analog design. Backpropagation, for example, is extremely sensitive to the steepness of the sigmoid function, and allowing for a variety of steepness levels in a fixed, analog implementation is challenging. Additionally, backpropagation performs poorly with a shallow sigmoid function. The requirement of a steep sigmoid function exacerbates analog range challenges, possibly making the implementation infeasible. RPROP tolerates a more shallow sigmoid activation steepness and performs consistently utilizing a constant activation steepness over all applications. Our RPROP-based, customized CDLM training phase requires 5000 training epochs, with the analog-based CDLM phase adding roughly 10% to the training time of the baseline training algorithm.

6. Evaluations

Cycle-accurate simulation and energy modeling. We use the MARSSx86 x86-64 cycle-accurate simulator [39] to model the performance of the processor. The processor is modeled after a single-core Intel Nehalem to evaluate the performance benefits of A-NPU acceleration over an aggressive out-of-order architecture[1]. We extended the simulator to include ISA-level support for A-NPU queue and dequeue instructions. We also augmented MARSSx86 with a cycle-accurate simulator for our A-NPU design and an 8-bit, fixed-point D-NPU with eight processing engines (PEs) as described in [18]. We use GCC v4.7.3 with -o3 to enable compiler optimization. The baseline in our experiments is the benchmark run solely on the processor without neural transformation. We use McPAT [33] for processor energy estimations. We model the energy of an 8-bit, fixed-point D-NPU using results from McPAT, CACTI 6.5 [37], and [22] to estimate its energy. Both the D-NPU and the processor operate at 3.4GHz at 0.9 V, while the A-NPU is clocked at one third of the digital clock frequency, 1.1GHz at 1.2 V, to achieve acceptable accuracy.

Circuit design for ANU. We built a detailed transistor-level SPICE model of the analog neuron, ANU. We designed and simulated the 8-bit, 8-input ANU in the Cadence Analog Design Environment using predictive technology models at 45 nm [6]. We ran detailed Spectre SPICE simulations to understand circuit behavior and measure ANU energy consumption. We used CACTI to estimate the energy of the A-NPU buffers. Evaluations consider all A-NPU components, both digital and analog. For the analog parts, we used direct measurements from the transistor-level SPICE simulations. For SRAM accesses, we used CACTI. We built an A-NPU cycle-accurate simulator to evaluate the performance improvements. Similar to McPAT, we combined simulation statistics with measurements from SPICE and CACTI to calculate A-NPU energy. To avoid biasing our study toward analog designs, all energy and performance comparisons are to an 8-bit, fixed-point D-NPU (8-bit inputs/weights/multiply-adders). For consistency with the available McPAT model for the baseline processor, we

[1]**Processor:** Fetch/Issue Width: 4/5, INT ALUs/FPUs: 6/6, Load/Store FUs: 1/1, ROB Entries: 128, Issue Queue Entries: 36, INT/FP Physical Registers: 256/256, Branch Predictor: Tournament 48 KB, BTB Sets/Ways: 1024/4, RAS Entries: 64, Load/Store Queue Entries: 48/48, Dependence Predictor: 4096-entry Bloom Filter, ITLB/DTLB Entries: 128/256 **L1:** 32 KB Instruction, 32 KB Data, Line Width: 64 bytes, 8-Way, Latency: 3 cycles **L2:** 256 KB, Line Width: 64 bytes, 8-Way, Latency: 6 cycles **L3:** 2 MB, Line Width 64 bytes, 16-Way, Latency: 27 cycles **Memory Latency:** 50 ns

Table 1: The evaluated benchmarks, characterization of each offloaded function, training data, and the trained neural network.

Benchmark Name	Description	Type	# of Function Calls	# of Loops	# of Ifs/ elses	# of x86-64 Instructions	Evaluation Input Set	Training Input Set	Neural Network Topology	Fully Digital NN MSE	Analog NN MSE (8-bit)	Application Error Metric	Fully Digital Error	Analog Error
blackscholes	Mathematical model of a financial market	Financial Analysis	5	0	5	309	4096 Data Point from PARSEC	16384 Data Point from PARSEC	6 -> 8 -> 8-> 1	0.000011	0.00228	Avg. Relative Error	6.02%	10.2%
fft	Radix-2 Cooley-Tukey fast fourier	Signal Processing	2	0	0	34	2048 Random Floating Point Numbers	32768 Random Floating Point Numbers	1 -> 4 -> 4 -> 2	0.00002	0.00194	Avg. Relative Error	2.75%	4.1%
inversek2j	Inverse kinematics for 2-joint arm	Robotics	4	0	0	100	10000 (x, y) Random Coordinates	10000 (x, y) Random Coordinates	2 -> 8 -> 2	0.000341	0.00467	Avg. Relative Error	6.2%	9.4%
jmeint	Triangle intersection detection	3D Gaming	32	0	23	1,079	10000 Random Pairs of 3D Triangle Coordinates	10000 Random Pairs of 3D Triangle Coordinate	18 -> 32 -> 8 -> 2	0.05235	0.06729	Miss Rate	17.68%	19.7%
jpeg	JPEG encoding	Compression	3	4	0	1,257	220x200-Pixel Color Image	Three 512x512-Pixel Color Images	64 -> 16 -> 8 -> 64	0.0000156	0.0000325	Image Diff	5.48%	8.4%
kmeans	K-means clustering	Machine Learning	1	0	0	26	220x200-Pixel Color Image	50000 Pairs of Random (r, g, b) Values	6 -> 8 -> 4 -> 1	0.00752	0.009589	Image Diff	3.21%	7.3%
sobel	Sobel edge detector	Image Processing	3	2	1	88	220x200-Pixel Color Image	One 512x512-Pixel Color Image	9 -> 8 -> 1	0.000782	0.00405	Image Diff	3.89%	5.2%

Table 2: Area estimates for the analog neuron (ANU).

Sub-circuit	Area
8×8-bit DAC	3,096 T*
8×Resistor Ladder (8-bit weights)	4,096 T + 1 KΩ (≈ 450T)
8×Differential Pair	48 T
I-to-V Resistors	20 KΩ (≈ 30 T)
Differential Amplifier	244 T
8-bit ADC	2550 T + 1 KΩ (≈ 450 T)
Total	≈ 10,964 T

*Transistor with width/length = 1

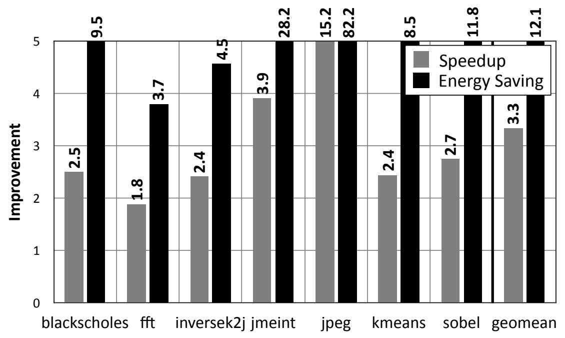


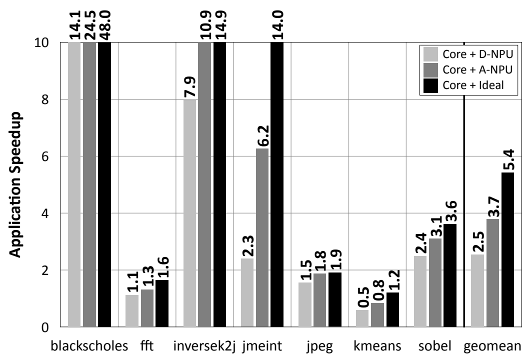
Figure 5: A-NPU with 8 ANUs vs. D-NPU with 8 PEs.

used McPAT and CACTI to estimate D-NPU energy. Even though we do not have a fabrication-ready layout for the design, in Table 2, we provide an estimate of the ANU area in terms of number of transistors. T denotes a transistor with $\frac{width}{length} = 1$. As shown, each ANU (which performs eight, 8-bit analog multiply-adds in parallel followed by a sigmoid) requires about 10,964 transistors. An equivalent digital neuron that performs eight, 8-bit multiply-adds and a sigmoid would require about 72,456 T from which 56,000 T are for the eight, 8-bit multiply-adds and 16,456 T for the sigmoid lookup. With the same compute capability, the analog neuron requires 6.6× fewer transistors than its equivalent digital implementation.

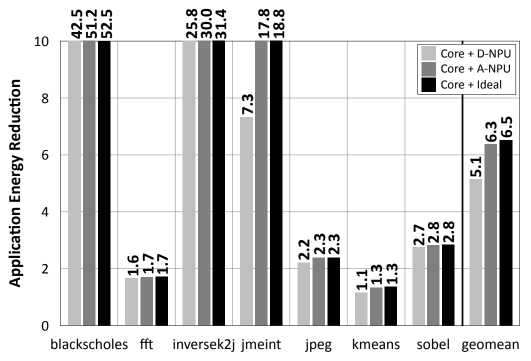
Benchmarks. We use the benchmarks in [18] and add one more, blackscholes. These benchmarks represent a diverse set of application domains, including financial analysis, signal processing, robotics, 3D gaming, compression, image processing. Table 1 summarizes information about each benchmark: application domain, target code, neural-network topology, training/test data and final application error levels for fully-digital neural networks and analog neural networks using our customized RPROP-based CDLM training algorithm. The neural networks were trained using either typical program inputs, such as sample images, or a limited number of random inputs. Accuracy results are reported using an independent data set, e.g, an input image that is different than the image used during training. Each benchmark requires an application-specific error metric, which is used in our evaluations. As shown in Table 1, each application benefits from a different neural network topology, so the ability to reconfigure the A-NPU is critical.

A-NPU vs 8-bit D-NPU. Figure 5 shows the average energy improvement and speedup for one invocation of an A-NPU over one invocation of an 8-bit D-NPU, where the A-NPU is clocked at $\frac{1}{3}$ the D-NPU frequency. On average, the A-NPU is 12.1× more energy efficient and 3.3× faster than the D-NPU. While consuming significantly less energy, the A-NPU can perform 64 multiply-adds in parallel, while the D-NPU can only perform eight. This energy-efficient, parallel computation explains why jpeg–with the largest neural network (64→16→8→64)–achieves the highest energy and performance improvements, 82.2× and 15.2×, respectively. The larger the network, the higher the benefits from A-NPU. Compared to a D-NPU, an A-NPU offers a higher level of parallelism with low energy cost that can potentially enable using larger neural networks to replace more complicated code.

Whole application speedup and energy savings. Figure 6 shows the whole application speedup and energy savings when the processor is augmented with an 8-bit, 8-PE D-NPU, our 8-ANU A-NPU, and an ideal NPU, which takes zero cycles and consumes zero energy. Figure 6c shows the percentage of dynamic instructions subsumed by the neural transformation of the candidate code. The results show, following the Amdahl's Law, that the larger the number of dynamic instructions subsumed, the larger the benefits from neural acceleration.



(a) Whole application speedup.



(b) Whole application energy saving.

	blackscholes	fft	inversek2j	jmeint	jpeg	kmeans	sobel
Percentage Instructions Subsumed	97.2%	67.4%	95.9%	95.1%	56.3%	29.7%	57.1%

(c) % dynamic instructions subsumed.

Figure 6: Whole application speedup and energy saving with D-NPU, A-NPU, and an Ideal NPU that consumes zero energy and takes zero cycles for neural computation.

Geometric mean speedup and energy savings with an A-NPU is 3.7× and 6.3× respectively, which is 48% and 24% better than an 8-bit, 8-PE NPU. Among the benchmarks, kmeans sees slow down with D-NPU and A-NPU-based acceleration. All benchmarks benefit in terms of energy. The speedup with A-NPU acceleration ranges from 0.8× to 24.5×. The energy savings range from 1.3× to 51.2×.As the results show, the savings with an A-NPU closely follows the ideal case, and, in terms of "energy", there is little value in designing a more sophisticated A-NPU. This result is due to the fact that the energy cost of executing instructions in the von Neumann, out-of-order pipeline is much higher than performing simple multiply-adds in the analog domain. Using physics laws (Ohm's law for multiplication and Kirchhoff's law for summation) and analog properties of devices to perform computation can lead to significant energy and performance benefits.

Application error. Table 3 shows the application-level errors with a floating point D-NPU, A-NPU with ideal sigmoid and our A-NPU which incorporates non-idealities of the analog sigmoid. Except for jmeint, which shows error above 10%, all of the applications show error less than or around 10%. Application average error rates with the A-NPU range from 4.1% to 10.2%. This quality-of-result loss is commensurate with other work on quality trade-offs. Among digital hardware approximation techniques, Truffle [17] and EnerJ [45] shows similar error (3–10%) for some applications and much greater error (above 80%) for others in a moderate configuration. Green [3] has error rates below 1% for some applications but greater than 20% for others. A case study [36] explores manual optimizations of the x264 video encoder that trade off 0.5–10% quality loss. As expected, the quality-of-results degradation with an A-NPU is more than a floating point D-NPU. However, the quality losses are commensurate with digital approximate computing techniques.

Table 3: Error with a floating point D-NPU, A-NPU with ideal sigmoid, and A-NPU with non-ideal sigmoid.

	blackscholes	fft	inversek2j	jmeint	jpeg	kmeans	sobel
Floating Point D-NPU	6.0%	2.7%	6.2%	17.6%	5.4%	3.2%	3.8%
A-NPU + Ideal Sigmoid	8.4%	3.0%	8.1%	18.4%	6.6%	6.1%	4.3%
A-NPU	10.2%	4.1%	9.4%	19.7%	8.4%	7.3%	5.2%

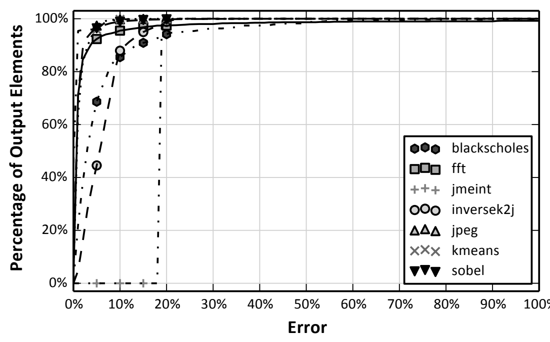


Figure 7: CDF plot of application output error. A point (x,y) indicates that y% of the output elements see error ≤ x%.

To study the application-level quality loss in more detail, Figure 7 illustrates the CDF (cumulative distribution function) plot of final error for each element of application's output. Each benchmark's output consists of a collection of elements– an image consists of pixels; a vector consists of scalars; etc. This CDF reveals the distribution of error among an application's output elements and shows that only a small fraction of the output elements see large quality loss with analog acceleration. The majority (80% to 100%) of each application's output elements have error less than 10% except for jmeint.

Exposing circuit limitations to the compiler. Figure 8 shows the effect of bit-width restrictions on application-level error, assuming 8 inputs per neuron. As the results suggest, exposing the bit-width limitations and the topology restrictions to the compiler enables our RPROP-based, customized CDLM training algorithm to find and train neural networks that can achieve accuracy levels commensurate with the digital approximation techniques, using only eight bits of precision for inputs, outputs, and weights, and eight inputs to the analog neurons. Several applications show less than 10% error even with fewer than eight bits. The results shows that there are many applications that can significantly benefit from analog

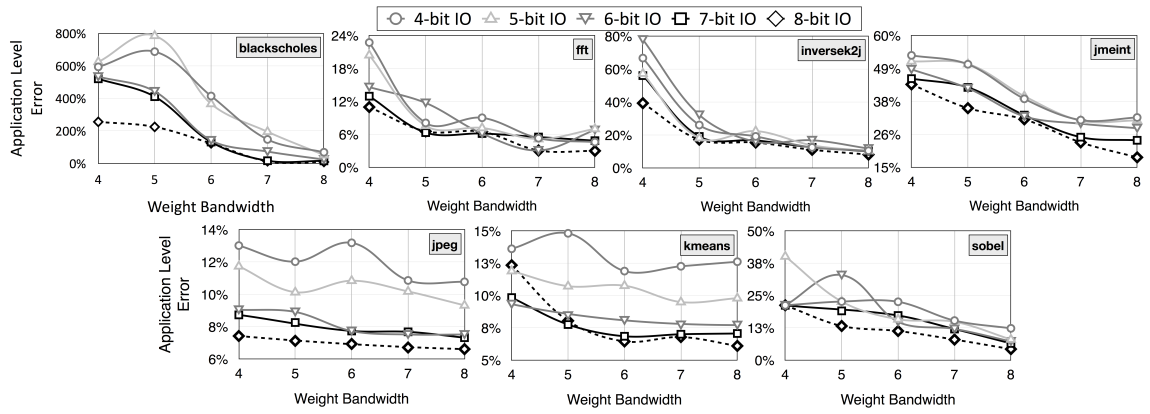
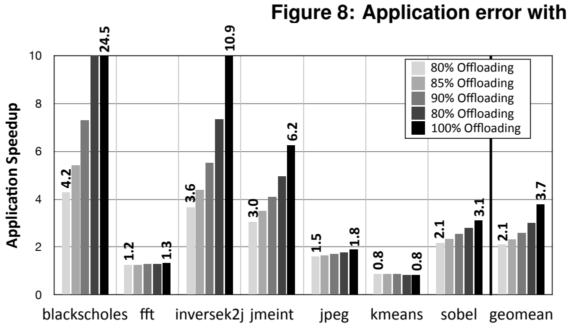
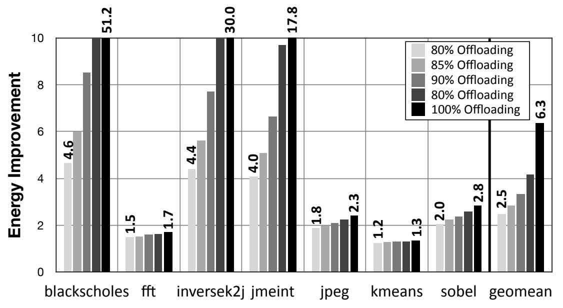


Figure 8: Application error with limited bit-width analog neural computation.



(a) Total application speedup with limited A-NPU invocations.



(b) Total application energy saving for limited A-NPU invocations.

Figure 9: Speedup/energy saving with limited A-NPU invocations.

acceleration without significant output quality loss.

Limited analog acceleration. We examine the effects on the benefits when, due to noise or pathological inputs, only a fraction of the invocations are offloaded to the A-NPU. In this case, the application falls back to the original code for the remaining invocations. Figure 9 depicts the application speedup and energy improvement when only 80%, 85%, 90%, 95%, and 100% of the invocations are offloaded to the A-NPU. The results suggest that even limited analog accelerators can provide significant energy and performance improvements.

7. Limitations and Considerations

Applicability. Not all applications can benefit from analog acceleration; however, our work shows that there are many that can. More rigorous optimization at the circuit level, as well as broadening the scope off application coverage by continued advancements at the neural transformation step, may provide significant improvements in accuracy and generality.

Other design points. This study evaluates the performance and energy improvements of an A-NPU assuming integration with a modern, high-performance processor. If low-power cores are used instead, we expect to see, and preliminary results confirm, that the performance benefits of an A-NPU increase, and that the energy benefits decrease.

Variability and noise. We designed the circuit with variability and noise as first-order concerns, and we made several design decisions to mitigate them. We limit both the input and weight bit widths, as well as the analog neuron input count to eight to provide quantization margins for variation/noise. We designed the sigmoid circuit one order of magnitude more shallow than the digital implementation to provide additional margins for variation and noise. We used a differential design, which provides resilience to noise by representing a value by the difference between two signals; as noise affects the pair of nearby signals similarly, the difference between the signals remains intact and the computation correct. Conversion to the digital domain after each analog neuron computation enforces computation integrity and reduces variation/noise susceptibility, while incurring energy and speed overheads. As mentioned in Section 6, to further improve the quality of the final result, we can refrain from A-NPU invocations and fall back to the original code as needed. An online noise-monitoring system could potentially limit the invocation of the A-NPU to low-noise situations. Incorporating a quantitative noise model into the training algorithm may improve robustness to analog noise.

Training for variability. A neural approach to approximate computing presents the opportunity to correct for certain types of analog-imposed inaccuracy, such as process variation, non-linearity, and other forms of non-ideality that are consistent for executions on a particular A-NPU hardware instance for some period of time. After an initial training phase that accounts for the predictable, compiler-exposed analog limitations, a second (shorter) training phase can adjust for hardware-specific non-idealities, sending training inputs and outputs to the A-NPU and adjusting network weights to minimize error. This correction technique is able to address inter and intra-chip process variation and hardware-dependent, non-ideal analog behavior.

Smaller technology nodes. This work is the start of using analog circuits for code acceleration. Providing benefits at smaller nodes may require using larger transistors for analog parts, trading off area for resilience. Energy-efficient performance is growing in importance relative to area efficiency, especially as CMOS scaling benefits continue to diminish.

8. Related Work

This research lies at the intersection of (a) general-purpose approximate computing, (b) accelerators, (c) analog and digital neural hardware, (d) neural-based code acceleration, (e) and limited-precision learning. This work combines techniques in all these areas to provide a compilation workflow and the architecture/circuit design that enables code acceleration with limited-precision mixed-signal neural hardware. In each area, we discuss the key related work that inspired our work.

General-purpose approximate computing. Several studies have shown that diverse classes of applications are tolerant to imprecise execution [20, 54, 34, 12, 45]. A growing body of work has explored relaxing the abstraction of full accuracy at the circuit and architecture level for gains in performance, energy, and resource utilization [13, 17, 44, 2, 35, 8, 28, 38, 18, 51, 46]. These circuit and architecture studies, although proven successful, are limited to purely digital techniques. We explore how a mixed-signal, analog-digital approach can go beyond what digital approximate techniques offer.

Accelerators. Research on accelerators seeks to synthesize efficient circuits or FPGA configurations to accelerate general-purpose code [41, 42, 11, 19, 29]. Similarly, static specialization has shown significant efficiency gains for irregular and legacy code [52, 53]. More recently, configurable accelerators have been proposed that allow the main CPU to offload certain code to a small, efficient structure [23, 24]. This paper extends the prior work on digital accelerators with a new class of mixed-signal, analog-digital accelerators.

Analog and digital neural hardware. There is an extensive body of work on hardware implementations of neural networks both in digital [40, 15, 55] and analog [5, 47, 49, 32, 48]. Recent work has proposed higher-level abstractions for implementation of neural networks [27]. Other work has examined fault-tolerant hardware neural networks [26, 50]. In particular, Temam [50] uses datasets from the UCI machine learning repository [21] to explore fault tolerance of a hardware neural network design. In contrast, our compilation, neural-network selection/training framework, and architecture design aim at applying neural networks to general-purpose code written in familiar programming models and languages, not explicitly written to utilize neural networks directly.

Neural-based code acceleration. A recent study [9] shows that a number of applications can be manually reimplemented with explicit use of various kinds of neural networks. That study did not prescribe a programming workflow, nor a preferred hardware architecture. More recent work exposes analog spiking neurons as primitive operators [4]. This work devises a new programming model that allows programmers to express digital signal-processing applications as a graph of analog neurons and automatically maps the expressed graph to a tiled analog, spiking-neural hardware. The work in [4] is restricted to the domain of applications whose inputs are real-world signals that should be encoded as pulses. Our approach addresses the long-standing challenges of using analog computation (programmability and generality) by not imposing domain-specific limitations, and by providing analog circuitry that is integrated with a conventional digital processor in a way that does not require a new programming paradigm.

Limited-precision learning. The work in [14] provides a complete survey of learning algorithms that consider limited precision neural hardware implementation. We tried various algorithms, but we found that CDLM [10] was the most effective. More sophisticated limited-precision learning techniques can improve the reported quality results in this paper and further confirm the feasibility and effectiveness of the mixed-signal, approach for neural-based code acceleration.

9. Conclusions

For decades, before the effective end of Dennard scaling, we consistently improved performance and efficiency while maintaining generality in general-purpose computing. As the benefits from scaling diminish, the community is facing an iron triangle; we can choose any two of performance, efficiency, and generality at the expense of the third. Solutions that improve performance and efficiency, while retaining as much generality as possible, are growing in importance. Analog circuits inherently trade accuracy for significant gains in energy-efficiency. However, it is challenging to utilize them in a way that is both programmable and generally useful. As this paper showed, the neural transformation of general-purpose approximable code provides an avenue for realizing the benefits of analog computation while targeting code written in conventional languages. This work provided an end-to-end solution for utilizing analog circuits for accelerating approximate applications, from circuits to compiler design. The insights from this work show that it is crucial to expose analog circuit characteristics to the compilation and neural network training phases. The NPU model offers a way to exploit analog efficiencies, despite their challenges, for a wider range of applications than is typically possible. Further, mixed-signal execution delivers much larger savings for NPUs than digital. However, this study is not conclusive. The full range of applications that can exploit mixed-signal NPUs is still unknown, as is whether it will be sufficiently large to drive adoption in high-volume microprocessors. It is still an open question how developers might reason about the acceptable level of error when an application undergoes an approximate execution including analog acceleration. Finally, in a noisy, high-performance microprocessor environment, it is unclear that an analog NPU would not be adversely affected. However, the significant gains from A-NPU acceleration and the diversity of the studied applications suggest a potentially promising path forward.

Acknowledgments

This work was supported in part by gifts from Microsoft Research and Google, NSF award CCF-1216611, and by C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

References

[1] P. E. Allen and D. R. Holberg, *CMOS Analog Circuit Design*. Oxford University Press, 2002.

[2] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE TC*, 2005.

[3] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *PLDI*, 2010.

[4] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, "Continuous real-world inputs can open up alternative accelerator designs," in *ISCA*, 2013.

[5] B. E. Boser, E. Säckinger, J. Bromley, Y. L. Cun, L. D. Jackel, and S. Member, "An analog neural network processor with programmable topology," *JSSC*, 1991.

[6] Y. Cao, "Predictive technology models," 2013. Available: http://ptm.asu.edu

[7] M. Carbin, S. Misailovic, and M. C. Rinard, "Verifying quantitative reliability for programs that execute on unreliable hardware," in *OOPSLA*, 2013.

[8] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, "Ultra-efficient (embedded) SOC architectures based on probabilistic CMOS (PCMOS) technology," in *DATE*, 2006.

[9] T. Chen, Y. Chen, M. Duranton, Q. Guo, A. Hashmi, M. Lipasti, A. Nere, S. Qiu, M. Sebag, and O. Temam, "BenchNN: On the broad potential application scope of hardware neural network accelerators," in *IISWC*, 2012.

[10] F. Choudry, E. Fiesler, A. Choudry, and H. J. Caulfield, "A weight discretization paradigm for optical neural networks," in *ICOE*, 1990.

[11] N. Clark, M. Kudlur, H. Park, S. Mahlke, and K. Flautner, "Application-specific processing on a general-purpose core via transparent instruction set customization," in *MICRO*, 2004.

[12] M. de Kruijf and K. Sankaralingam, "Exploring the synergy of emerging workloads and silicon reliability trends," in *SELSE*, 2009.

[13] M. de Kruijf, S. Nomura, and K. Sankaralingam, "Relax: An architectural framework for software recovery of hardware faults," in *ISCA*, 2010.

[14] S. Draghici, "On the capabilities of neural networks using limited precision weights," *Elsevier NN*, 2002.

[15] H. Esmaeilzadeh, P. Saeedi, B. N. Araabi, C. Lucas, and S. M. Fakhraie, "Neural network stream processing core (NnSP) for embedded systems," in *ISCAS*, 2006.

[16] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *ISCA*, 2011.

[17] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ASPLOS*, 2012.

[18] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," in *MICRO*, 2012.

[19] K. Fan, M. Kudlur, G. Dasika, and S. Mahlke, "Bridging the computation gap between programmable processors and hardwired accelerators," in *HPCA*, 2009.

[20] Y. Fang, H. Li, and X. Li, "A fault criticality evaluation framework of digital systems for error tolerant video applications," in *ATS*, 2011.

[21] A. Frank and A. Asuncion, "UCI machine learning repository," 2010. Available: http://archive.ics.uci.edu/ml

[22] S. Galal and M. Horowitz, "Energy-efficient floating-point unit design," *IEEE TC*, 2011.

[23] V. Govindaraju, C. Ho, and K. Sankaralingam, "Dynamically specialized datapaths for energy efficient computing," in *HPCA*, 2011.

[24] S. Gupta, S. Feng, A. Ansari, S. Mahlke, and D. August, "Bundled execution of recurring traces for energy-efficient general purpose processing," in *MICRO*, 2011.

[25] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, 2011.

[26] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microarchitectures," in *ISCA*, 2011.

[27] A. Hashmi, A. Nere, J. J. Thomas, and M. Lipasti, "A case for neuromorphic ISAs," in *ASPLOS*, 2011.

[28] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *ISLPED*, 1999.

[29] Y. Huang, P. Ienne, O. Temam, Y. Chen, and C. Wu, "Elastic cgras," in *FPGA*, 2013.

[30] C. Igel and M. Hüsken, "Improving the RPROP learning algorithm," in *NC*, 2000.

[31] D. A. Johns and K. Martin, *Analog Integrated Circuit Design*. John Wiley and Sons, Inc., 1997.

[32] A. Joubert, B. Belhadj, O. Temam, and R. Héliot, "Hardware spiking neurons design: Analog or digital?" in *IJCNN*, 2012.

[33] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *MICRO*, 2009.

[34] X. Li and D. Yeung, "Exploiting soft computing for increased fault tolerance," in *ASGI*, 2006.

[35] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," in *ASPLOS*, 2011.

[36] S. Misailovic, S. Sidiroglou, H. Hoffman, and M. Rinard, "Quality of service profiling," in *ICSE*, 2010.

[37] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *MICRO*, 2007.

[38] S. Narayanan, J. Sartori, R. Kumar, and D. L. Jones, "Scalable stochastic processors," in *DATE*, 2010.

[39] A. Patel, F. Afram, S. Chen, and K. Ghose, "MARSSx86: A full system simulator for x86 CPUs," in *DAC*, 2011.

[40] K. W. Przytula and V. K. P. Kumar, Eds., *Parallel Digital Implementations of Neural Networks*. Prentice Hall, 1993.

[41] A. Putnam, D. Bennett, E. Dellinger, J. Mason, P. Sundararajan, and S. Eggers, "CHiMPS: A high-level compilation flow for hybrid CPU-FPGA architectures," in *FPGA*, 2008.

[42] R. Razdan and M. D. Smith, "A high-performance microarchitecture with hardware-programmable functional units," in *MICRO*, 1994.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press, 1986.

[44] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, "Sage: Self-tuning approximation for graphics engines," in *MICRO*, 2013.

[45] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *PLDI*, 2011.

[46] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," in *MICRO*, 2013.

[47] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *IJCNN*, 2008.

[48] R. St. Amant, D. A. Jiménez, and D. Burger, "Mixed-signal approximate computation: A neural predictor case study," *IEEE MICRO Top Picks*, vol. 29, no. 1, January/February 2009.

[49] S. M. Tam, B. Gupta, H. A. Castro, and M. Holler, "Learning on an analog VLSI neural network chip," in *SMC*, 1990.

[50] O. Temam, "A defect-tolerant accelerator for emerging high-performance applications," in *ISCA*, 2012.

[51] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality-programmable vector processors for approximate computing," in *MICRO*, 2013.

[52] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor, "Conservation cores: Reducing the energy of mature computations," in *ASPLOS*, 2010.

[53] G. Venkatesh, J. Sampson, N. Goulding, S. K. Venkata, M. Taylor, and S. Swanson, "QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores," in *MICRO*, 2011.

[54] V. Wong and M. Horowitz, "Soft error resilience of probabilistic inference applications," in *SELSE*, 2006.

[55] J. Zhu and P. Sutton, "FPGA implementations of neural networks: A survey of a decade of progress," in *FPL*, 2003.