Newton:

Gravitating Towards the Physical Limits of Crossbar Acceleration

Anirban Nag, Rajeev Balasubramonian, Vivek Srikumar, Ross Walker
University of Utah

Ali Shafiee
Samsung

John Paul Strachan
Hewlett Packard Enterprise

Naveen Muralimanohar
Amazon

Many recent works take advantage of highly parallel analog in-situ computation in memristor crossbars to accelerate the many vector-matrix multiplication operations in deep neural networks (DNNs). However, these in-situ accelerators have two significant shortcomings: The ADCs account for a large fraction of chip power and area, and these accelerators adopt a homogeneous design in which every resource is provisioned for the worst case. By addressing both problems, the new architecture, called Newton, moves closer to achieving optimal energy per neuron for crossbar accelerators. We introduce new techniques that apply at different levels of the tile hierarchy, some leveraging heterogeneity and others relying on divide-and-conquer numeric algorithms to reduce computations and ADC pressure. Finally, we place constraints on how a workload is mapped to tiles, thus helping reduce resource-provisioning in tiles. For many convolutional-neural-network (CNN) dataflows and structures, Newton achieves a 77-percent decrease in power, 51-percent improvement in energy-efficiency, and 2.1× higher throughput/area, relative to the state-of-the-art In-Situ Analog Arithmetic in Crossbars (ISAAC) accelerator.

The last two years have seen a flurry of activity in designing machine-learning accelerators targeted at enterprise servers, self-driving cars, and mobile devices. Similar to our work, most of these recent works have focused on inference in artificial neural networks that achieve state-of-the-art accuracies on challenging image-classification workloads.

While most of these recent accelerators have used digital architectures, a few have leveraged analog acceleration on memristor crossbars that take advantage of in-situ computation to dramatically reduce data-movement costs. Each crossbar is assigned to execute parts of the neural-

Published by the IEEE Computer Society
0272-1732/18/$33.00 ©2018 IEEE

network computation and programmed with the corresponding weight values. Input neuron values are fed to the crossbar, and—by leveraging Kirchhoff's law—the crossbar outputs the corresponding dot product. The neuron output undergoes analog-to-digital conversion before being sent to the next layer. Multiple small-scale prototypes of this approach have been demonstrated.[1]

The design constraints for digital accelerators are very different from analog ones. High communication overhead and the memory bottleneck are first-order design constraints in digital, whereas the computation overhead arising from analog-to-digital conversions and balancing the extent of digital computation are more critical in analog accelerators. In this work, we show that computation is a critical problem in analog. We leverage numeric algorithms to reduce conversion overheads. We also introduce "just-right" resource provisioning to efficiently handle the common case.

With these innovations in place, our new design, called Newton, moves the analog architecture closer to the bare minimum amount of energy required to process one neuron. We define an ideal neuron as one that keeps the weight in place adjacent to a digital ALU, retrieves the input from an adjacent single-row eDRAM unit, and—after performing one digital operation—writes the result to another adjacent single-row eDRAM unit. This energy is lower than that for a similarly ideal analog neuron because of the ADC cost. This ideal neuron operation consumes 0.33 pJ. An average DaDianNao operation consumes 3.5 pJ because it pays a high price in data movement for inputs and weights.[2] An average ISAAC operation consumes 1.8 pJ because it pays a moderate price in data movement for inputs (weights are in situ) and a high price for ADC.[3] An average Eyeriss operation consumes 1.67 pJ because of an improved dataflow to maximize reuse.[4] The innovations in Newton push the analog architecture closer to the ideal neuron by consuming 0.85 pJ per operation.

BACKGROUND

Analog Accelerators

Two CNN accelerators introduced in the recent past, ISAAC and PRIME,[3,5] have leveraged memristor crossbars to perform dot-product operations in the analog domain. We focus on ISAAC here because it out-performs PRIME in terms of throughput, accuracy, and ability to handle signed values. ISAAC is also able to achieve nearly 8× higher throughput than the digital accelerator DaDianNao.

The ISAAC Pipeline

In ISAAC, memristive crossbar arrays are used to perform analog dot-product operations. Neuron inputs are provided as voltages to wordlines, neuron weights are represented by pre-programmed cell conductances, and neuron outputs are represented by the currents in each bitline. The neuron outputs are processed by an ADC and shift-and-add circuits. They are then sent as inputs to the next layer of neurons. As shown in Figure 1(a), ISAAC is a tiled architecture, meaning one or more tiles are dedicated to process one layer of the neural network. To perform inference for one input image, neuron outputs are propagated from tile to tile until all network layers have been processed.

An ISAAC chip consists of many tiles connected in a mesh topology. Each tile includes an eDRAM buffer that supplies inputs to in-situ multiply-accumulate (IMA) units. The IMA units consist of memristor crossbars (which perform the dot-product computation), ADCs, and shift-and-add circuits that accumulate the digitized results. With a design space exploration, the tile is provisioned with an optimal number of IMAs, crossbars, ADCs, and so on. Within a crossbar, a 16-bit weight is stored with 2 bits per cell, across eight columns. A 16-bit input is supplied as voltages over 16 cycles, 1 bit per cycle, using a trivial DAC array. The partial outputs are shifted and added across eight columns and across 16 cycles to give the output of many parallel multiply-accumulate (MAC) operations. Thus, there are two levels of pipelining in ISAAC: (i) the intra-tile pipeline (where inputs are read from eDRAM, processed by crossbars in 16 cycles, and

aggregated) and (ii) the inter-tile pipeline (where neuron outputs are transferred from one layer to the next). The intra-tile pipeline has a cycle time of 100 ns, matching the latency for a crossbar read. Inputs are sent to a crossbar in an IMA using an input HTree network. The input HTree has sufficient bandwidth to keep all crossbars active without bubbles. Each crossbar has a dedicated ADC operating at 1.28 GSamples/s shared across its 128 bitlines to convert the analog output to digital in 100 ns. An HTree network is then used to collect digitized outputs from crossbars.

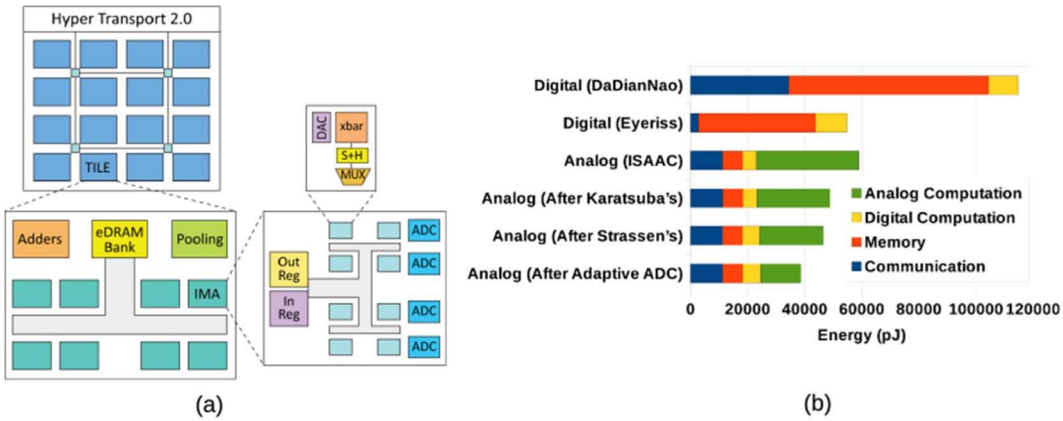

Figure 1. (a) The ISAAC architecture. (b) Energy breakdown of vector-matrix multiplication in existing digital and analog pipelines for the proposed optimizations.

Even though an analog architecture consists of both digital and analog computations, the overhead of analog dominates with 61 percent of the total power.[3] Consider a 1×128 vector being multiplied with a 128×128 matrix (all values are 16 bits). Figure 1(b) shows the energy breakdown of the vector-matrix multiplication pipeline compared against digital designs for various architectures. As the figure shows, communication and memory accesses are the major limiting factor for digital architectures—whereas for analog, computation overhead dominates, primarily arising from the ADC costs.

PROPOSAL

Intra-IMA Optimizations

HTree modifications

ISAAC did not place any constraints on how a neural network can be mapped to its many tiles and IMAs. As a result, its resources, notably the HTrees within the IMA, are provisioned to handle the worst case. If multiple layers were to share an IMA, we would need multiple HTrees to support the simultaneous aggregation of multiple neurons. This introduces a non-trivial area overhead and offers a flexibility that is rarely useful to the workloads we examined. We modify mapping by placing the constraint that an IMA cannot be shared by multiple network layers. While this inflexibility can waste a few resources, we observe that it also significantly reduces the HTree size and hence area per IMA. The architecture is still general-purpose, meaning arbitrary CNNs can be mapped to it.

ISAAC was agnostic to how a single synaptic weight was scattered across multiple bitlines; it therefore requires a 16-bit wide HTree. Instead, we adopt the following approach to boost area-efficiency. A 16-bit weight is scattered across eight 2-bit cells; each cell is placed in a different crossbar. We also embed the shift-and-add units in the HTree, as shown in Figure 2(b). So, the shift-and-add unit at the leaf of the HTree adds the digitized 9-bit dot-product results emerging from two neighboring crossbars. Because the operation is a shift-and-add, it produces an 11-bit

result. The next shift-and-add unit takes two 11-bit inputs to produce a 13-bit result, and so on. This leads to an efficient pipeline and lowers the HTree widths.

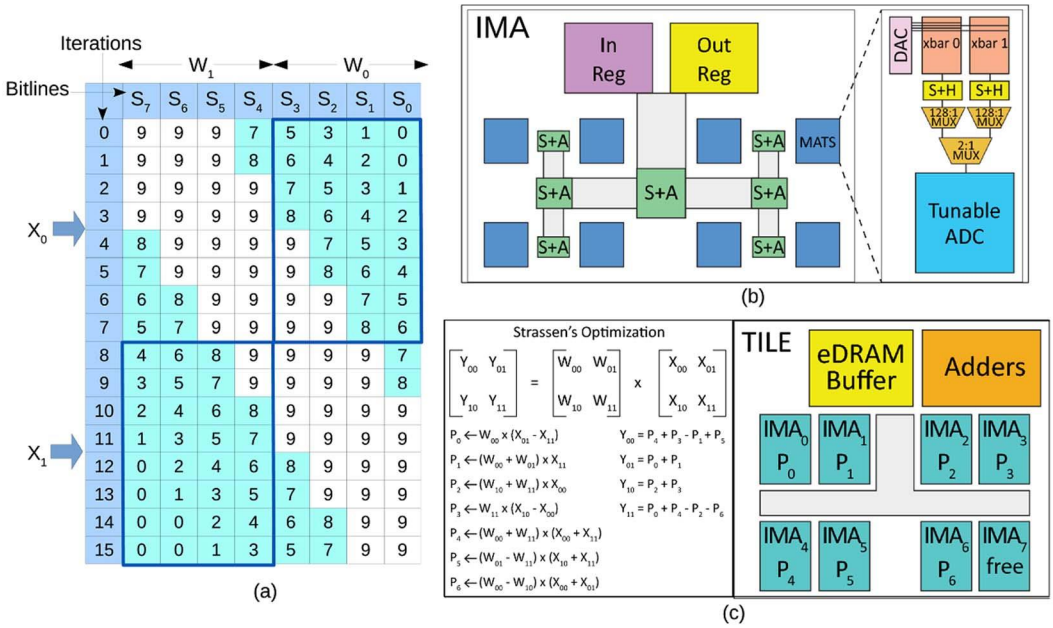

Figure 2. (a) Heterogeneous ADC sampling resolution, (b) IMA supporting Karatsuba's algorithm, and (c) mapping Strassen's algorithm to a tile.

Adaptive ADCs

A simple dot-product operation on 16-bit values performed using crossbars typically results in an output of more than 16 bits. In the example discussed earlier, using 2-bit cells in crossbars and 1-bit DACs yields a 39-bit output. Once the scaling factor is applied, the least significant 10 bits are dropped. The most significant 13 bits represent an overflow that cannot be captured in the 16-bit result, so they are effectively used to clamp the result to a maximum value.

What is of note here is that the output from every crossbar column in every iteration is being resolved with a high-precision 9-bit ADC, but many of these bits contribute to either the 10 least significant bits or the 13 most significant bits that are eventually going to be ignored. This is an opportunity to lower the ADC precision and ignore some bits, depending on the column and the iteration being processed. Figure 2(a) shows the number of relevant bits emerging from every column in every iteration. Note that before dropping the highest-ignored least significant bit, we use rounding modes to generate carries.[6]

The ADC accounts for a significant fraction of IMA power. When the ADC is operating at a lower resolution, it has less work to do. In every 100-ns iteration, we tune the resolution of a successive approximation (SAR) ADC to match the requirement in Figure 2(a). The ADC simply gates off its circuits until the next sample is provided.[7] Thus, the use of adaptive ADCs helps reduce IMA power while having no impact on performance. We are also ignoring bits that do not show up in a 16-bit fixed-point result, so we are not impacting the functional behavior of the algorithm, thus having zero impact on algorithm accuracy. It is also worth pointing out that the adaptive ADC technique is compatible with the encoding used by ISAAC to reduce ADC resolution by 1 bit.

Karatsuba's divide-and-conquer multiplication technique

Karatsuba's divide-and-conquer algorithm (see Equation 1) manages to reduce the complexity of multiplying two n-bit numbers $O(n^2)$ to $O(n^{1.5})$ by dividing the numbers into two halves of $n/2$

bits; instead of performing four $n/2$-bit multiplications, it calculates the result with two $n/2$-bit multiplications and one $(n/2+1)$-bit multiplication.

$$W = 2^{N/2}W_1 + W_0$$
$$X = 2^{N/2}X_1 + X_0 \quad (1)$$
$$WX = 2^{N}W_1X_1 + 2^{N/2}(W_1X_0 + W_0X_1) + W_0X_0 = (2^{N} - 2^{N/2})W_1X_1 + 2^{N/2}(W_1 + W_0)(X_1 + X_0) + (1 - 2^{N/2})W_0X_0$$

To illustrate the benefit of this technique, consider the same example discussed earlier using 128×128 crossbars, 2-bit cells, and a 1-bit DAC. The product of input X and weight W is performed on eight crossbars in 16 cycles. In Equation 1, W_0X_0 is performed on four crossbars in eight iterations (since we are dealing with fewer bits for weights and inputs). The same is true for W_1X_1. A third set of crossbars stores the weights ($W_1 + W_0$) and receives the pre-computed inputs ($X_1 + X_0$). This computation is spread across five crossbars and nine iterations. We see that the total amount of work has reduced by 15 percent.

There are a few drawbacks, as well. A computation now takes 17 iterations instead of 16. The net area also increases because the network must send inputs X_0 and X_1 in parallel, an additional crossbar is needed, the output buffer is larger to store sub-products, and 128 1-bit full adders are required to compute ($X_1 + X_0$). Again, given that the ADC is the primary bottleneck, these other overheads are relatively minor.

To implement Karatsuba's algorithm, we modify the IMA, as shown in Figure 2(b). The changes are localized to a single mat. Each mat now has two crossbars that share the DAC and ADC. Given the size of the ADC, the extra crossbar per mat has a minimal impact on area. The left crossbars in four of the mats now store W_0, the left crossbars in the other four mats store W_1, the right crossbars in five of the mats store $W_0 + W_1$, and the right crossbars in three of the mats are unused. In the first eight iterations, the eight ADCs are used by the left crossbars. In the next nine iterations, five ADCs are used by the right crossbars. As discussed earlier, the main objective here is to lower power by reducing use of the ADC. Divide-and-conquer can be recursively applied further. When applied again, the computation keeps eight ADCs busy in the first four iterations and six ADCs busy in the next ten iterations. This is a 28-percent reduction in ADC use and a 13-percent reduction in execution time. But, there is an area penalty because 20 crossbars are needed per IMA.

Intra-Tile Optimizations

The previous sub-section focused on techniques to improve an IMA; we now shift our focus to the design of a tile. We first use a divide-and-conquer approach at the tile level. We then create heterogeneous tiles that suit convolutional and fully connected layers.

Strassen's algorithm

A divide-and-conquer approach can also be applied to matrix-matrix multiplication. Note that all of our computations are vector-matrix multiplications. But when a layer is replicated, multiple input vectors are being fed to the same matrix of weights; so, replicated layer computations are matrix-matrix multiplications. By partitioning each matrix X and W into four sub-matrices, we can express matrix-matrix multiplication in terms of multiplications of sub-matrices (see Figure 2(c)). A typical algorithm would require eight sub-matrix multiplications, followed by an aggregation step. Linear algebra manipulations can perform the same computation with seven sub-matrix multiplications (P_0–P_6) with appropriate pre- and post-processing. Figure 2(c) shows the mapping of computations within a tile to implement Strassen's algorithm. The computations (P_0–P_6) in Strassen's algorithm are mapped to seven IMAs in the tile, thus freeing up the eighth IMA and reducing ADC usage.

The two divide-and-conquer optimizations reduce the computational energy by 20.6 percent. However, they have very little impact on other digital accelerators. For example, these algorithms may impact the efficiency of the neuron functional units (NFUs) in DaDianNao, but DaDianNao area is dominated by eDRAM banks, not NFUs. On the other hand, analog computations

are dominated by ADCs, so efficient computation does noticeably impact overall efficiency. Furthermore, some of the preprocessing for these algorithms is performed when installing weights on analog crossbars, but it has to be performed "on the fly" for digital accelerators. The computation re-factoring due to these techniques needs additional adder blocks in the pipeline, incurring a storage overhead of 4.3 percent and a latency overhead of 4.5 percent.

Different tiles for convolutions and classifiers

While ISAAC uses the same homogeneous tile for the entire chip, we observe that convolutional layers have very different resource demands than fully connected classifier layers. The classifier (or FC) layer has to aggregate a set of inputs required by a set of crossbars. The crossbars then perform their computation; the inputs are discarded and a new set of inputs is aggregated. This results in the following properties for the classifier layer:

- The classifier layer has a high communication-to-compute ratio, so the router bandwidth puts a limit on how often the crossbars can be busy.
- The classifier also has the highest synaptic weight requirement because every neuron has private weights.
- The classifier has low buffering requirements; an input is seen by several neurons in parallel, and the input can be discarded right after.

We therefore design special tiles customized for classifier layers that:

- have a higher crossbar-to-ADC ratio (4:1 instead of 1:1),
- operate the ADC at a lower rate (10 MSamples/s instead of 1.2 GSamples/s), and
- have a smaller eDRAM buffer size (4 Kbytes instead of 16 Kbytes).

Storage-efficient FC tiles reduce inter-tile communication and hence the communication energy. For small-scale workloads that are trying to fit on a single chip, we propose a chip in which some of the tiles are conv tiles and some are classifier tiles (a ratio of 1:1 is a good fit for most of our workloads). For large-scale workloads that use multiple chips, each chip can be homogeneous; we use roughly an equal number of conv chips and classifier chips. The results consider both cases. While a digital accelerator such as ScaleDeep uses heterogeneity to manage the byte-fetch-to-flops ratio in different layers, Newton employs new knobs to manage varying ADC utilization rates.

METHODOLOGY

For modeling the energy and area of the eDRAM buffers and on-chip interconnect like the HTree and tile bus, we use CACTI 6.5 at 32 nm. The area and energy model of a memristor crossbar is based on its experimental realization by Miao Hu and colleagues.[8] We adapt the area and energy of a 1-bit DAC, shift-and-add circuits, max/average pooling block, sigmoid operation, and router similar to the analysis in ISAAC.[3] We avail the same HyperTransport serial link model for off-chip interconnects as used by DaDianNao and ISAAC.[2,3] While our buffers can also be implemented with SRAM, we use eDRAM to make an apples-to-apples comparison with the ISAAC baseline. Newton is only used for inference with a delay of 16.4 ms to pre-load weights in a chip. The Newton architecture uses the same 8-bit ADC at 32 nm as used in ISAAC, partly because it yields the best configuration in terms of area/power and meets the sampling frequency requirement, and partly because it can be reconfigured for different resolutions.[9] This ADC is equipped with a memory and state logic to realize the control logic of Figure 2(a) to tune the ADC, which requires no extra time.

The ADC power for different sampling resolutions is modeled by gating off the other components except the sampling clock. We create an analytic model for a Newton pipeline within an IMA and within a tile, and then map the suite of benchmarks, making sure that there are no structural hazards in any of these pipelines. We consider network bandwidth limitations in our simulation model to estimate throughput. Since ISAAC is a throughput architecture, we perform

an iso-throughput comparison of the Newton architecture with ISAAC for the different intra-IMA or intra-tile optimizations. Since the dataflow in the architecture is bounded by the router bandwidth, we allocate enough resources until the network saturates to create our baseline model. Similar to ISAAC, data transfers between tiles on chip and on the HT link across chips have been statically routed to make it conflict-free. Like ISAAC, the latency and throughput of Newton for the given benchmarks can be calculated analytically using a deterministic execution model. Since there aren't any runtime dependencies on the control flow or dataflow of the deep networks, analytical estimates are enough to capture the behavior of cycle-accurate simulations.

As with any new technology, a memristor crossbar has unique challenges, mainly in two respects. First, mapping a matrix onto a memristor crossbar array requires programming (or writing) cells with the highest precision possible. Second, real circuits deviate from ideal operation due to parasitics such as wire resistance, device variation, and write/read noise. All of these factors can cause the actual output to deviate from its ideal value. Recent work has captured many of these details to show the viability of prototypes to tolerate errors.[1,8] The adaptive ADC technique leads to a reduced output precision requirement, which helps the cause of tolerating noise, whereas the other techniques are modifications in the digital domain.

RESULTS

We use ISAAC as the baseline architecture and augment it with the proposed optimizations. This work considers a suite of state-of-the-art workloads representative of various dataflows and network sizes.

We explored Newton's design space and chose a moderately sized IMA that processes 128 inputs for 256 neurons and has high computational-efficiency and low crossbar under-utilization (9 percent), given the mapping constraints.[7] In Figure 3(a), we show the improvement of Newton over ISAAC for the various workloads. The improvement in area-efficiency is largely due to the HTree modification that decreases the number of wires and heterogeneous tiles that make the FC tiles storage-efficient. Karatsuba's technique comes at the cost of a 6.4-percent reduction in area-efficiency because of the need for more crossbars and the increase in HTree bandwidth to send the sum of inputs. The overall improvement in area is 60 percent.

The ADCs contribute 49 percent of the chip power in ISAAC. Optimizing the number of ADC conversions and their precision leads to large gains in energy-efficiency (adaptive ADC: 13 percent, Karatsuba's: 17 percent, and Strassen's: 4.5 percent). Divide-and-conquer can be recursively applied further, but experiments show that applying it once is nearly as good as applying it twice, and much less complex. The heterogeneous tiles are able to bring down the peak power requirement by 25 percent, as the ADCs in the FC tile sample at a much lower frequency. We also observe that the above techniques are not sensitive to the ADC design; using a recently published neuromorphic architecture, friendly ADC leads to 30-percent improvement in energy-efficiency but increases the area of the chip by 35 percent.[10]

Figure 3(b) plots the incremental effect of each of our techniques on peak computational-efficiency (GOPS/s/mm^2) and power-efficiency (GOPS/s/W) of DaDianNao, ISAAC, and Newton. We see that both adaptive ADCs and the divide-and-conquer approach play a significant role in increasing the power-efficiency. While the impact of Strassen's technique is not visible in this graph, it manages to free up resources (one in every eight IMAs) in a tile, thus providing room for more compact mapping of networks and reduced ADC utilization.

Figure 3(c) shows an iso-area comparison of the 8-bit version of Newton with Google's tensor processing unit (TPU) architecture.[11] For the TPU, we perform enough batch processing to not exceed the latency target of 7 ms as demanded by most application developers. Newton's latency is comfortably less than 7 ms for all the evaluated benchmarks. We also model the TPU with GDDR5 memory to allocate sufficient bandwidth.

Newton exhibits 10.3× improvement in throughput even though its peak computational-efficiency is 12.3× better. This is because, when operating on an FC layer, some crossbars in Newton remain idle. While Newton is only 1.6× better than the TPU in terms of peak power-efficiency, the actual benefit goes up for real workloads, increasing to 3.4×. Some large networks

like MSRA3 require a lower batching degree for the TPU to meet the latency target; this increases the memory pressure and causes idling in compute units. From the figure, it can also be noted that the throughput improvement of AlexNet and ResNet aren't as high as the other benchmarks because of their relatively smaller networks. This increases the batch size, improving the data locality for FC layer weights in the TPU.

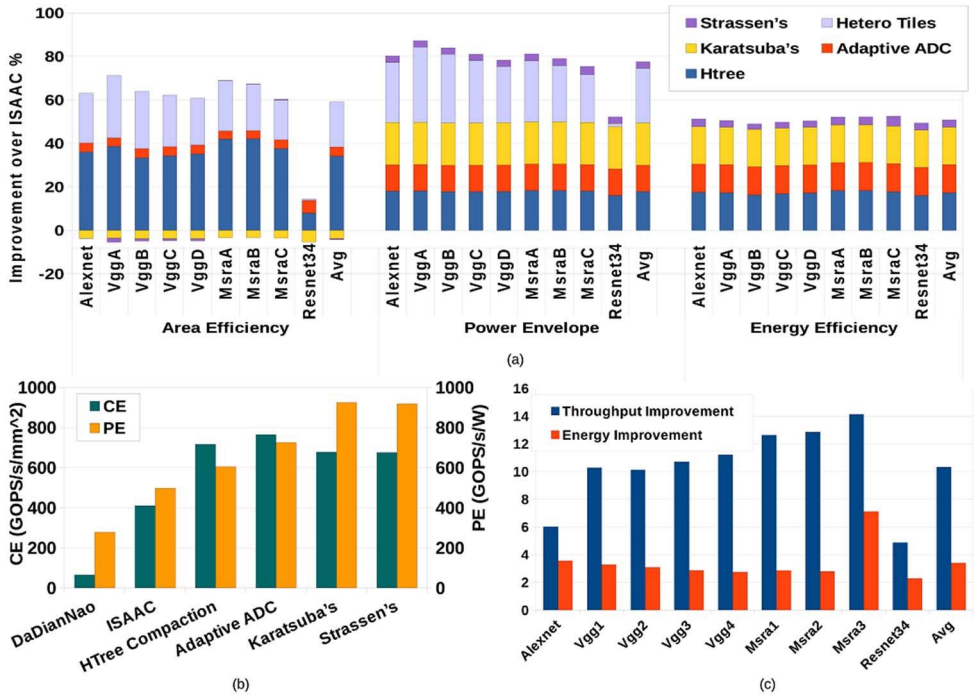

Figure 3. (a) Breakdown of improvement over ISAAC on area-efficiency, power envelope, and energy-efficiency due to different techniques. (b) Peak computational-efficiency (CE) and power-efficiency (PE) metrics of different schemes, along with baseline digital and analog accelerators. (c) Comparison with Google's TPU.[11]

CONCLUSION

In this work, we target resource provisioning and efficiency in a crossbar-based DNN accelerator. Starting with the ISAAC architecture, we show that three approaches—heterogeneity, mapping constraints, and divide-and-conquer—can be applied within a tile and within an IMA. This results in a smaller HTree, energy-efficient ADCs with varying resolution, energy- and area-efficiency in classifier layers, and fewer computations. While some of these techniques are used in software libraries, our key finding is that they have little impact on digital accelerators that are more memory-bound, but have a significant impact on analog accelerators that are ADC-bound. Many of these ideas would also apply to a general accelerator for matrix-matrix multiplication, as well as to other neural networks such as RNN and long short-term memory (LSTM). The Newton architecture cuts the current gap between ISAAC and an ideal neuron in half.

REFERENCES

1. M. Hu et al., "Memristor-Based Analog Computation and Neural Network Classification with a Dot Product Engine," *Wiley-VCH Advanced Materials*, 2018.
2. Y. Chen et al., "DaDianNao: A Machine-Learning Supercomputer," *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

3. A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.
4. Y.H. Chen, J.E. Emer, and V.S. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.
5. P. Chi et al., "PRIME: A Novel Processing-in-Memory Architecture for Neural Network Computation in ReRAM-Based Main Memory," *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.
6. S. Gupta et al., "Deep learning with limited numerical precision," *ICML'15 Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 2015.
7. A. Nag et al., "Newton: Gravitating Towards the Physical Limits of Crossbar Acceleration," *arXiv*, 2018.
8. M. Hu et al., "Dot-Product Engine for Neuromorphic Computing: Programming 1T1M Crossbar to Accelerate Matrix-Vector Multiplication," *53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.
9. L. Kull et al., "A 3.1 mW 8b 1.2 GS/s Single-Channel Asynchronous SAR ADC With Alternate Comparators for Enhanced Speed in 32 nm Digital SOI CMOS," *IEEE Journal of Solid-State Circuits*, vol. 48, 2013.
10. L. Danial et al., "Breaking Through the Speed-Power-Accuracy Tradeoff in ADCs using a Memristive Neuromorphic Architecture," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.
11. N. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," *ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

ABOUT THE AUTHORS

Anirban Nag is a PhD student in the School of Computing at the University of Utah. His research focuses on accelerators for machine learning and genomics. Contact him at anirban@cs.utah.edu.

Rajeev Balasubramonian is a professor in the School of Computing at the University of Utah. He has a PhD from the University of Rochester. His research focuses on memory systems, security, and accelerators. He is a senior member of the IEEE. Contact him at rajeev@cs.utah.edu.

Vivek Srikumar is an assistant professor in the School of Computing at the University of Utah. His research spans the areas of machine learning, natural learning processing, and artificial intelligence. He has a PhD from the University of Illinois at Urbana-Champaign, and he was a post-doctoral scholar at Stanford University. Contact him at svivek@cs.utah.edu.

Ross Walker is an assistant professor in the Department of Electrical and Computer Engineering at the University of Utah. He has a PhD in electrical engineering from Stanford University. His research interests include electronic circuits and systems for neuroscience and neural engineering applications. He serves on the IEEE Biomedical and Life Science Circuits and Systems Technical Committee. Contact him at ross.walker@utah.edu.

Ali Shafiee is a senior hardware engineer at Samsung. His research interests include deep-learning architecture, memory systems, and memory security. He has a PhD in computer science from the University of Utah. Contact him at ali.shafiee@samsung.com.

John Paul Strachan is an HPE Master Technologist and leads the Rebooting Computing team at Hewlett Packard Labs. Key areas of interest are computing applications in machine learning, network security, and optimization problem solving. He has a PhD in applied physics from Stanford University. Contact him at john-paul.strachan@hpe.com.

Naveen Muralimanohar is a software architect at Amazon. He has a PhD from the University of Utah. His research areas include emerging memory technologies, accelerators, and distributed systems. Contact him at naveen.murali@gmail.com.