BIT-PRAGMATIC DEEP NEURAL NETWORK COMPUTING

Jorge Albericio , Patric Judd, Alberto Delmas Lascorz, Sayeh Sharify & Andreas Moshovos
Electrical and Computer Engineering
University of Toronto
Toronto, ON, M5S 3G4, Canada
{jorge, juddpatr, delmasl1,sayeh,moshovos}@ece.utoronto.ca

ABSTRACT

We quantify a source of ineffectual computations when processing the multiplications of the convolutional layers in Deep Neural Networks (DNNs) and propose *Pragmatic* (*PRA*), an architecture that exploits it improving performance and energy efficiency. The source of these ineffectual computations is best understood in the context of conventional multipliers which generate internally multiple *terms*, that is, products of the multiplicand and powers of two, which added together produce the final product Wallace (1964). At runtime, many of these terms are zero as they are generated when the multiplicand is combined with the zero-bits of the multiplicator. While conventional bit-parallel multipliers calculate all terms in parallel to reduce individual product latency, *PRA* calculates only the non-zero terms resulting in a design whose execution time for convolutional layers is ideally proportional to the number of activation bits that are 1. Measurements demonstrate that for the convolutional layers on Convolutional Neural Networks and during inference, *PRA* improves performance by 4.3x over the DaDiaNao (DaDN) accelerator Chen et al. (2014) and by 4.5x when DaDN uses an 8-bit quantized representation Warden (2016). *DaDN* was reported to be 300x faster than commodity graphics processors.

1 INTRODUCTION

Deep Neural Network (DNN) hardware typically uses either 16-bit fixed-point Chen et al. (2014) or quantized 8-bit numbers Warden (2016) and bit-parallel compute units. For convolutional layers, that account for most of the execution time in Convolutional Neural Networks (CNNs) during image classification, these bit-parallel engines perform many ineffectual computations. Specifically, these layers perform several several inner products, where multiple pairs of weights and activations are multiplied and then reduced into an output activation. Any time a zero bit of an activation or a weight is multiplied it adds nothing to the final output activations. These ineffectual bits are introduced by the conventional positional number representation and if avoided it would take even less time to calculate each product improving energy and performance. As a first step, this work targets the ineffectual bits of activations only. Section 2 shows that in recent image classification networks 93% and 69% of activation bit and weight products are ineffectual when using respectively 16-bit fixed-point and 8-bit quantized representations.

This work presents *Pragmatic* (*PRA*) a DNN accelerator whose goal is to process only the *essential* (non-zero) bits of the input activations *PRA* employs the following four key techniques: 1) on-the-fly conversion of activations from a storage representation (e.g., conventional positional number or quantized) into an explicit representation of the essential bits only, 2) bit-serial activation/bit-parallel weight processing, an idea borrowed from *STR* Judd et al. (2016b;a) but adapted for the aforementioned representation, 3) judicious SIMD (single instruction multiple data) lane grouping to maintain wide memory accesses and to avoid fragmenting and enlarging the multi-MB on-chip weight memories (Sections 5 and 5.1), and 4) computation re-arrangement (Section 5.1) to reduce datapath area. All evaluated *PRA* variants maintain wide memory accesses and use highly-parallel SIMD-style (single-instruction multiple-data) computational units. *PRA* introduces an additional dimension upon which software can improve performance and energy efficiency by controlling ac-

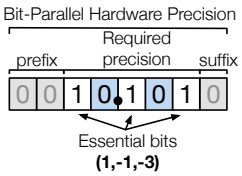


Figure 1: Sources of ineffectual computation with conventional positional representation and fixed-length hardware precision.

tivation values judiciously in order to reduce their essential bit content while maintaining accuracy. This work explores such an alternative, where the software explicitly communicates how many prefix and suffix bits to discard after each layer.

Experimental measurements with recent CNNs for image classification demonstrate that most straightforward *PRA* variant, boosts average performance for the convolutional layers to 2.59x over the state-of-the-art *DaDN* accelerator. *Pragmatic*'s average energy efficiency is 1.48x over *DaDN* and its area overhead is 1.35x. Another variant further boosts performance to 3.1x over *DaDN* at the expense of an additional 0.7% area.

2 MOTIVATION

Let us assume a p-bit bit-parallel multiplier using a straightforward implementation of the "Shift and Add" algorithm where $n \times s$ is calculated as $\sum_{i=0}^{p} n_i \cdot (s \ll i)$, where n_i the i-*th* bit of n. The multiplier computes p *terms*, each a product of s and of a bit of n, and adds them to produce the final result. The terms and their sum can be calculated concurrently to reduce latency Wallace (1964).

With such a hardware arrangement there are two sources of ineffectual computations that result from: 1) an *Excess of Precision* (EoP), and 2) *Lack of Explicitness* (LoE). Figure 1 shows an example illustrating these sources with a bit-parallel multiplier using an 8-bit unsigned fixed-point number with 4 fractional and 4 integer bits. While $10.101_{(2)}$ requires just five bits, our 8-bit bit-parallel multiplier will zero-extend it with two prefix and one suffix bits. This is an example of EoP and is due to the fixed-precision hardware. Two additional ineffectual bits appear at positions 1 and -2 as a result of LoE in the positional number representation. In total, five ineffectual bits will be processed generating five ineffectual terms.

Our number could be represented with an explicit list of its three constituent powers of 2: (1,-1,-3). While such a representation may require more bits and thus be undesirable for storage, coupled with the abundant parallelism that is present in DNNs layers, it provides an opportunity to revisit hardware design improving performance and energy efficiency.

Table 5 reports the *essential bit content* of the activation stream of recent CNNs for two commonly used fixed length representations: 1) 16-bit fixed-point of DaDianNao Chen et al. (2014), 2) 8-bit quantized of Tensorflow Warden (2016). The essential bit content is the average number of non-zero bits that are 1. Two measurements are presented per representation: over all neuron values ("All"), and over the non-zero neurons ("NZ") as accelerators that can skip zero activations for fixed-point representations have been recently proposed Han et al. (2016); Albericio et al. (2016).

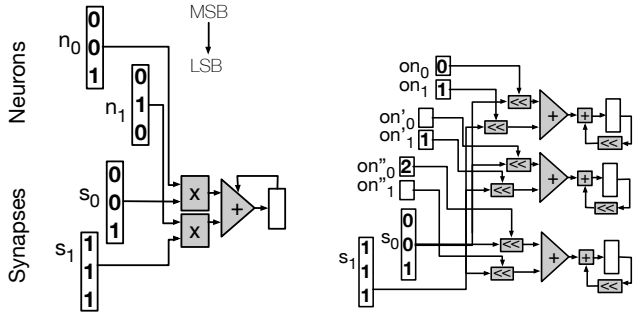
When considering all activations, the essential bit-content is at most 12.7% and 38.4% for the fixed-point and the quantized representations respectively. Even when considering the non-zero activations the essential bit content remains well below 50% suggesting that the potential exists to improve performance and energy efficiency over approaches that target zero valued activations only.

3 *Pragmatic*: A SIMPLIFIED EXAMPLE

This section illustrates the idea behind *Pragmatic* via a simplified example.

	Alexnet	NiN	Google	VGGM	VGGS	VGG19
16-bit Fixed-Point						
All	7.8%	10.4%	6.4%	5.1%	5.7%	12.7%
NZ	18.1%	22.1%	19.0%	16.5%	16.7%	24.2%
8-bit Quantized						
All	31.4%	27.1%	26.8%	38.4%	34.3%	16.5%
NZ	44.3%	37.4%	42.6%	47.4%	46.0%	29.1%

Table 1: Average fraction of non-zero bits per activation for two fixed-length representations: 16-bit fixed-point, and 8-bit quantized. **All:** over all activations. **NZ:** over non-zero activation only.



(a) Bit-Parallel Unit (b) *Pragmatic* Unit

Figure 2: An Example Illustrating How *Pragmatic* Skips Ineffectual Activation Bits Yet Exceeding the Performance of a Bit-Parallel Engine

The bit-parallel unit of Figure 2a multiplies two activations with their respective weights and via an adder reduces the two products. The unit reads *all* activation and weight, $(n_0 = 001_{(2)}, n_1 = 010_{(2)})$ and $(s_0 = 001_{(2)}, s_1 = 111_{(2)})$ respectively in a single cycle. As a result, the two sources of inefficiency EoP and LoE manifest here: n_0 and n_1 are represented using 3 bits instead of 2 respectively due to EoP. Even in 2 bits, they each contain a zero bit due to LoE. As a result, four ineffectual terms are processed when using standard multipliers such as those derived from the Shift and Add algorithm. In general, given N activation and weight pairs, this unit will take $\lceil N/2 \rceil$ cycles to process them regardless of their precision and the essential bit content of the activations.

Figure 2b shows a simplified *PRA* engine. In this example, activations are no longer represented as vectors of bits but as vectors of offsets of the essential bits. For example, activation $n_0 = 001_{(2)}$ is represented as $on_0 = (0)$, and a activation value of $111_{(2)}$ would be represented as $(2, 1, 0)$. An out-of-band bit (wire) not shown indicates the activation's end. A shifter per activation uses the offsets to effectively multiply the corresponding weight with the respective power of 2 before passing it to the adder tree. As a result, *PRA* processes only the non-zero terms avoiding all ineffectual computations that were due to EoP or LoE. To match the throughput of the bit-parallel engine of Figure 2a, we take advantage of weight reuse and processes multiple activations groups in parallel. In this example, six activations $(n_0 = 001_{(2)}, n_1 = 010_{(2)}, n'_0 = 000_{(2)}, n'_1 = 010_{(2)}, n''_0 = 010_{(2)}, n''_1 = 000_{(2)})$ are combined with the two weights as shown. For this example, *PRA* would process the six activation and weight pairs in a single cycle, a speedup of $3\times$ over the bit-parallel engine.

4 BASELINE SYSTEM: DADIANNAO

Pragmatic is demonstrated as a modification of the *DaDianNao* accelerator (*DaDN*) proposed by Chen *et al.* Chen et al. (2014). Figure 3a shows a *DaDN* tile which processes 16 filters concurrently calculating 16 activation and weight products per filter for a total of 256 products per cycle. To do, each cycle the tile accepts 16 weights per filter for total of 256 weight, and 16 input activations. The tile multiplies each weight with only one activation whereas each activation is multiplied with 16 weight, one per filter. The tile reduces the 16 products into a single partial output activation per filter, for a total of 16 partial output activations for the tile. Each *DaDN* chip comprises 16 such tiles, each

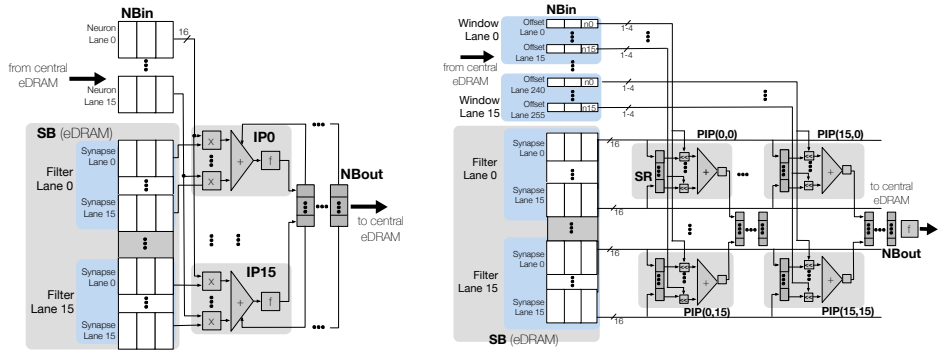


Figure 3: a) DaDianNao Tile. b) Pragmatic Tile.

processing a different set of 16 filters per cycle. Accordingly, each cycle, the whole chip processes 16 activations and $256 \times 16 = 4K$ weights producing $16 \times 16 = 256$ partial output activations.

Internally, each tile has: 1) a synapse buffer (SB) that provides 256 weights per cycle one per synapse lane, 2) an input neuron buffer[1] (NBin) which provides 16 activations per cycle through 16 neuron lanes, and 3) a neuron output buffer (NBout) which accepts 16 partial output activations per cycle. In the tile's datapath, or the *Neural Functional Unit* (NFU) each neuron lane is paired with 16 synapse lanes one from each filter. Each synapse and neuron lane pair feed a multiplier and an adder tree per filter lane reduces the 16 per filter products into a partial sum. In all, the filter lanes produce each a partial sum per cycle, for a total of 16 partial output activations per NFU. Once a full window is processed, the 16 resulting sums, are fed through a non-linear activation function, f, to produce the 16 final output activations. The multiplications and reductions needed per cycle are implemented via 256 multipliers one per synapse lane and sixteen 17-input (16 products plus the partial sum from NBout) adder trees one per filter lane.

DaDN's main goal was minimizing off-chip bandwidth while maximizing on-chip compute utilization. To avoid fetching weights from off-chip, *DaDN* uses a 2MB eDRAM SB per tile for a total of 32MB eDRAM. All inter-layer activations except for the initial input and the final output are stored in a 4MB shared central eDRAM *Neuron Memory* (NM) which is connected via a broadcast interconnect to the 16 NBin buffers. Off-chip accesses are needed only for reading the input image, the filter weights once per layer, and for writing the final output.

Terminology: For clarity, in what follows $n(x, y, i)$ and $o(x, y, i)$ refer to an input and an output activation at coordinates (x, y, i) respectively. The weight of filter f at coordinates (x, y, i) is denoted as $s^f(x, y, i)$. The term *brick* refers to a set of 16 elements of a 3D activation or weight array which are contiguous along the i dimension, e.g., $n(x, y, i)...n(x, y, i + 15)$. Bricks will be denoted by their origin element with a B subscript, e.g., $n_B(x, y, i)$. The term *pallet* refers to a set of 16 bricks corresponding to adjacent, using a stride S, windows along the x or y dimensions, e.g., $n_B(x, y, i)...n_B(x, y + 15 \times S, i)$ and will be denoted as $n_P(x, y, i)$. The number of activations per brick, and bricks per pallet are design parameters.

Processing Approach: Processing starts by reading from external memory the first layer's weights synapses, and the input image. The weights are distributed over the SBs and the input is stored into NM. Each cycle an input activation brick is broadcast to all units. Each units reads 16 weight bricks from its SB and produces a partial output activation brick which it stores in its NBout. Once computed, the output activations are stored through NBout to NM and then fed back through the NBins when processing the next layer. Loading the next set of activations from external memory can be overlapped with the processing of the current layer as necessary.

[1]Chen et al. (2014) used the terms *neuron* and *synapse* to refer to activations and weights respectively and named the various components accordingly. We maintain this terminology for the design's components.

5 *Pragmatic*

PRA's goal is to process only the essential bits of the activations. To do so *PRA* a) converts, on-the-fly, the input activation representation into one containing only the essential bits, and b) processes one essential bit per activation and a full 16-bit weight per cycle. Since *PRA* processes activation bits serially, it may take up to 16 cycles to produce a product of a activation and a weight. To always match or exceed the performance of the bit-parallel units of *DaDN*, *PRA* processes more activations concurrently exploiting the abundant parallelism of the convolutional layers. The remaining of this section describes in turn: 1) an appropriate activation representation, 2) the way *PRA* calculates terms, 3) how multiple terms are processed concurrently to maintain performance on par with *DaDN* in the worst case, and 4) how *PRA*'s units are supplied with the necessary activations from NM.

Input Activation Representation: *PRA* starts with an input activation representation where it is straightforward to identify the next essential bit each cycle. One such representation is an explicit list of *oneffsets*, that is of the constituent powers of two. For example, an activation $n = 5.5_{(10)} = 0101.1_{(2)}$ would be represented as $n = (2, 0, -1)$. In the implementation described herein, activations are stored in 16-bit fixed-point in NM, and converted on-the-fly in the *PRA* representation as they are broadcast to the tiles. A single oneffset is processed per activation per cycle. Each oneffset is represented as (pow, eon) where pow is a 4-bit value and eon a single bit which if set indicates the end of a activation. For example, $n = 101_{(2)}$ is represented as $n^{PRA} = ((0010, 0)(0000, 1))$.

Calculating a (weight, activation) product: *PRA* calculates the product of weight s and activation n as:

$$s \times n = \sum_{\forall f \in n^{PRA}} s \times 2^f = \sum_{\forall f \in n^{PRA}} (n \ll f)$$

That is, each cycle, the weight s multiplied by f, the next constituent power two of n, and the result is accumulated. This multiplication can be implemented as a shift and an AND.

Boosting Compute Bandwidth over *DaDN*: To match *DaDN*'s performance *PRA* needs to process the same number of effectual terms per cycle. Each *DaDN* tile calculates 256 activation and weight products per cycle, or $256 \times 16 = 4K$ terms. While most of these terms will be in practice ineffectual, to guarantee that *PRA* always performs as well as *DaDN* it should process $4K$ terms per cycle. For the time being let us assume that all activations contain the same number of essential bits, so that when processing multiple activations in parallel, all units complete at the same time and thus can proceed with the next set of activations in sync. The next section will relax this constraint.

Since *PRA* processes activations bits serially, it produces one term per activation bit and weight pair and thus needs to process $4K$ such pairs concurrently. The choice of which $4K$ activation bit and weight pairs to process concurrently can adversely affect complexity and performance. For example, it could force an increase in SB capacity and width, or an increase in NM width, or be ineffective due to unit underutilization given the commonly used layer sizes.

Fortunately, it is possible to avoid increasing the capacity and the width of the SB and the NM while keeping the units utilized as in *DaDN*. Specifically, a *PRA* tile can read 16 weight bricks and the equivalent of 256 activation bits as *DaDN*'s tiles do (*DaDN* processes 16 16-bit activations or 256 activation bits per cycle). Specifically, as in *DaDN*, each *PRA* tile processes 16 weight bricks concurrently, one per filter. However, differently than *DaDN* where the 16 weight bricks are combined with just one activation brick which is processed bit-parallel, *PRA* combines each weight brick with 16 activation bricks, one from each of 16 windows, which are processed bit-serially. The same 16 activation bricks are combined with all weight bricks. These activation bricks form a *pallet* enabling the same weight brick to be combined with all. For example, in a single cycle a *PRA* title processing filters 0 through 15 could combine combine $s_B^0(x, y, 0), ..., s_B^15(x, y, 0)$ with $n_B^{PRA}(x, y, 0), n_B^{PRA}(x+2, y, 0), ...n_B^{PRA}(x+31, y, 0)$ assuming a layer with a stride of 2. In this case, $s^4(x, y, 2)$ would be paired with $n^{PRA}(x, y, 2)$, $n^{PRA}(x + 2, y, 2)$, ..., $n^{PRA}(x + 31, y, 2)$ to produce the output weights $on(x, y, 4)$ through $on(x + 15, y, 4)$.

As the example illustrates, this approach allows each weight to be combined with one activation per window whereas in *DaDN* each weight is combined with one activation only. In total, 256 essential activation bits are processed per cycle and given that there are 256 weights and 16 windows, *PRA*

processes $256 \times 16 = 4K$ activation bit and weight pairs, or terms per cycle producing 256 partial output activations, 16 per filter, or 16 partial output activation bricks per cycle.

Supplying the Inputs: Thus far it was assumed that all input activations have the same number of essential bits. Under this assumption, all neuron lanes complete processing their terms at the same time, allowing *PRA* to move on to the next activation pallet and the next set of weight bricks in one step. This allows *PRA* to reuse *STR*'s approach for fetching the next pallet from the single-ported NM Judd et al. (2016b;a). Briefly, with unit stride the 256 weights would be typically all stored in the same NM row or at most over two adjacent NM rows and thus can be fetched in at most two cycles. When the stride is more than one, the weights will be spread over multiple rows and thus multiple cycles will be needed to fetch them all. Fortunately, fetching the next pallet can be overlapped with processing the current one. Accordingly, if it takes NM_C to access the next pallet from NM, while the current pallet requires P_C cycles to process, the next pallet will begin processing after $max(NM_C, P_C)$ cycles. When $NM_C > P_C$ performance is lost waiting for NM.

In practice it highly unlikely that all activations will have the same number of essential bits. In general, each neuron lane if left unrestricted will advance at a different rate. In the worst case, each neuron lane may end up needing activations from a different activation brick, thus breaking *PRA*'s ability to reuse the same weight brick. This is undesirable if not impractical as it would require partitioning and replicating the SB so that 4K unrelated weight could be read per cycle, and it would also increase NM complexity and bandwidth.

Fortunately, these complexities can be avoided with *pallet-level neuron lane synchronization* where all neuron lanes "wait" (a neuron lane that has detected the end of its activation forces zero terms while waiting) for the one with the most essential bits to finish before proceeding with the next pallet. Under this approach it does not matter which bits are essential per activation, only how many exist. Since, it is unlikely that most pallets will contain an activation with 16 essential terms, *PRA* will improve performance over *DaDN*. Section 5.1 will discuss finer-grain synchronization schemes that lead to even better performance. Before doing so, however, we detail *PRA*'s design.

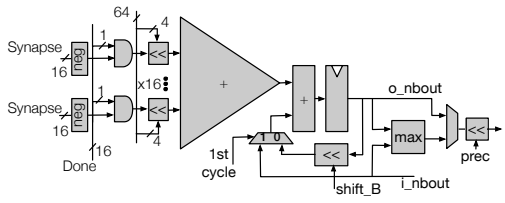


Figure 4: Pragmatic Inner Product Unit.

5.1 Structure and Performance and Area Optimizations

Figure 3b shows the *Pragmatic* tile architecture which comprises an array of $16 \times 16 = 256$ *pragmatic inner product units (PIPs)*. PIP(i,j) processes an activation oneffset from the i-th window and its corresponding weight from the j-th filter. Specifically, all the PIPs along the i-th row receive the same weight brick belonging to the i-th filter and all PIPs along the j-th column receive an oneffset from each activation from one activation brick belonging to the j-th window. The necessary activa-

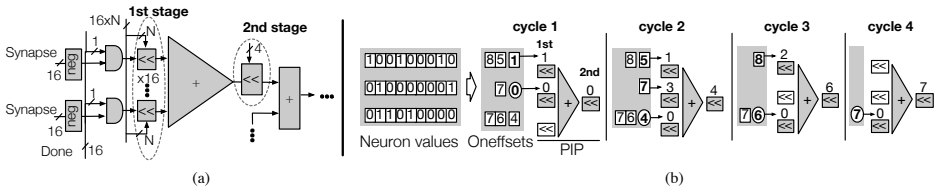


Figure 5: 2-stage shifting. a) Modified PIP. b) Example: Processing three 9-bit weight and activation pairs with $L = 2$.

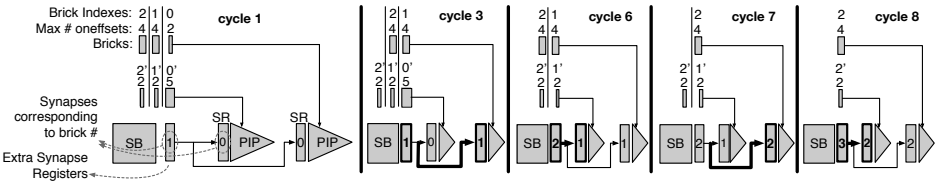


Figure 6: Per-column synchronization example: one extra synapse register and 1x2 PIP array capable of processing two windows in parallel. The two numbers per brick show: the first from the top is the brick's index, $(0, 1, 2)$ and $(0', 1', 2')$ for the bricks of the first and second window. The second is the maximum count of oneffsets in its activations, $(2, 4, 4)$ and $(5, 2, 2)$ respectively. The numbers in the registers indicate the index of the corresponding bricks, i.e., a synapse register containing a K stores the weights corresponding to activation bricks with indexes K and K'. In cycles 3 to 8, thicker lines indicate registers being loaded or wires being used.

tion oneffsets are read from NBin where they have been placed by the Dispatcher and the Oneffset generators units as Section 5.1 explains. Every cycle NBin sends 256 oneffsets 16 per window lane. All the PIPs in a column receive the same 16 oneffsets, corresponding to the activations of a single window. When the tile starts to process a new activation pallet, 256 weights are read from SB through its 256 synapse lanes as in *DaDN* and are stored in the synapse registers (SR) of each PIP. The weights and oneffsets are then processed by the PIPs.

Pragmatic Inner-Product Unit: Figure 4 shows the PIP internals. Every cycle, 16 weights are combined with their corresponding oneffsets. Each oneffsets controls a shifter effectively multiplying the weight with a power of two. The shifted weights are reduced via the adder tree. An AND gate per weight supports the injection of a null terms when necessary. In the most straightforward design, the oneffsets use 4-bits, each shifter accepts a 16-bit weight and can shift it by up to 15 bit positions producing a 31-bit output. Finally, the adder tree accepts 31-bit inputs. Section 5.1 presents an enhanced design that requires narrower components improving area and energy.

Dispatcher and Oneffset Generators The *Dispatcher* reads 16 activation bricks from NM, as expected by the *PRA* tiles. The *oneffset generator* converts their activations on-the-fly to the oneffset representation, and broadcasts one oneffset per activation per cycle for a total of 256 oneffsets to all titles. Fetching and assembling the 16 activation bricks from NM is akin to fetching words with a stride of S from a cache structure. Once the 16 activation bricks have been collected, 256 oneffset generators operate in parallel to locate and communicate the next oneffset per activation. A straightforward 16-bit leading one detector is sufficient. The latency of the oneffset generators and the dispatcher can be readily hidden as they can be pipelined as desired overlapping them with processing in the *PRA* tiles.

Reducing Title Area with 2-Stage Shifting: Any shift can be performed in two stages as two smaller shifts: $a \ll K = a \ll (K' + C) = ((a \ll K') \ll C)$. Thus, to shift and add T weights by different offsets $K_0, ..., K_T$, we can decompose the offsets into sums with a common term C, e.g., $K_i = K'_i + C$. Accordingly, PIP processing can be rearranged using a two stage processing where the first stage uses a per weight specific offset K'_i, and the second stage, the common across all weights offset C. This arrangement can be used to reduce the width of the weight shifters and of the adder tree by sharing one common shifter after the adder tree as Figure 5a shows. A design parameter, L, defines the number of bits controlling the weight shifters so that the design can process oneffsets which differ by less than 2^L in a single cycle. This reduces the size of the weight shifters and reduces the size of the adder tree to support terms of $16 + 2^L - 1$ bits only.

Increasing Performance with Per-Column Neuron Lane Synchronization: The pallet neuron lane synchronization scheme of Section 5 is one of many possible synchronization schemes. Finergrain neuron lane synchronization schemes are possible leading to higher performance albeit at a cost. Among them, *per column* neuron lane synchronization is an appealing scheme offering a good balance of cost vs. performance. Here each PIP column operates independently but all the PIPs along the same column synchronize before moving to the next activation brick. Since the PIPs along the same column operate in sync, they all process one set of 16 weight bricks which can be read using the existing SB interface. However, given that different PIP columns operate now out-of-

sync, the SB would be accessed more frequently and could become a bottleneck. There are two concerns: 1) different PIP columns may need to perform two independent SB reads while there are only one SB port and one common bus connecting the PIP array to the SB, and 2) there will be repeat accesses to SB that will increase SB energy, while the SB is already a major consumer of energy. These concerns are addressed as follows: 1) only one SB access can proceed per cycle thus a PIP column may need to wait when collisions occur. 2) A set of registers, or *synapse set registers* (SSRs) are introduced in front of the SB each holding a recently read set of 16 weight bricks. Since all PIP columns will eventually need the same set of weight bricks, temporarily buffering them avoids fetching them repeatedly from the SB. Once a weight set has been read into an SSR, it stays there until all PIP columns have copied it (a 4-bit down counter is sufficient for tracking how many PIP columns have yet to read the weight set). This policy guarantees that the SB is accessed the same number of times as in *DaDN*. However, stalls may incur as a PIP column has to be able to store a new set of weights into an SSR when it reads it from the SB. Figure 6 shows an example. Since each neuron lane advances independently, in the worst case, the dispatcher may need to fetch 16 independent activation bricks each from a different pallet. The Dispatcher can buffer those pallets to avoid rereading NM, which would, at worst, require a 256 pallet buffer. However, given that the number SSRs restricts how far apart the PIP columns can be, and since Section 6.2 shows that only one SSR is sufficient, a two pallet buffer in the dispatcher is all that is needed.

Further Increasing Performance with Improved Oneffset Encoding: Since PIPs in *Pragmatic* can negate any input term, it is possible to enhance the oneffset generator to generate fewer oneffsets for neuron values containing runs of ones by allowing signed oneffsets Booth (1951).

This improved generator reduces runs of adjacent oneffsets $a...b$ into pairs of the form $a+1, -b$. Single oneffsets or gaps inside runs are represented by a positive or negative oneffset, respectively. For example a neuron value of 11011 that would normally be encoded with oneffsets $(4, 3, 1, 0)$ can instead be represented with $(5, -3, +2, -0)$ or even more economically with $(5, -2, -0)$. This is equivalent to a Radix-4 Booth encoding and will never emit more than $\left\lfloor \frac{x}{2} + 1 \right\rfloor$ oneffsets, where x is the neuron precision.

This encoding will never produce more oneffsets compared to the baseline encoding. However, because of the 2-stage shifting, it is possible that this encoding will increase the number of cycles needed. This will happen when the oneffset distribution among the bit groups being processed together during 2-stage shifting changes.

Finally, booth encoding is conventionally used to reduce the number of cycles needed to perform multiplication in single shift-and-add multipliers typically reserved for low cost low performance designs, or to reduce the depth of bit-parallel multipliers. *Pragmatic* with its 2-stage shifting and judicious lane synchronization enables its practical use in a massively data-parallel accelerator boosting performance beyond what is possible with bit-parallel units.

The Role of Software: *PRA* enables an additional dimension upon which hardware and software can attempt to further boost performance and energy efficiency, that of controlling the essential activation value content. This work investigates a software guided approach where the precision requirements of each layer are used to zero out a number of prefix and suffix bits at the output of each layer. Using the profiling method of Judd *et al.,* Judd et al. (2015), software communicates the precisions needed by each layer as meta-data. The hardware trims the output activations before writing them to NM using AND gates and precision derived bit masks.

6 EVALUATION

The performance, area and energy efficiency of *Pragmatic* is compared against *DaDN* Chen et al. (2014) and *Stripes* Judd et al. (2016b), two state-of-the-art DNN accelerators. *DaDN* is the fastest bit-parallel accelerator proposed to date that processes all activations regardless of theirs values, and *STR* improves upon *DaDN* by exploiting the per layer precision requirements of DNNs. *Cnvlutin* improves upon *DaDN* by skipping most zero- or near-zero-valued activations Albericio et al. (2016), however, *Stripes* has been shown to outperform it.

After reviewing the experimental methodology the rest of this section is organized as follows: Sections 6.1 and 6.2 explore the *PRA* design space considering respectively single- and 2-stage shifting configurations, and column synchronization. Section 6.2 reports energy efficiency for the best

Network	Per Layer Activation Precision in Bits
AlexNet	9-8-5-5-7
NiN	8-8-8-9-7-8-8-9-9-8-8-8
GoogLeNet	10-8-10-9-8-10-9-8-9-10-7
VGG_M	7-7-7-8-7
VGG_S	7-8-9-7-9
VGG_19	12-12-12-11-12-10-11-11-13-12-13-13-13-13-13-13

Table 2: Per convolutional layer activation precision profiles.

configuration. Section 6.4 analyzes the contribution of the software provided precisions. Finally, Section 6.5 reports performance for designs using an 8-bit quantized representation.

Methodology: The same methodology is used for all systems for consistency. A custom cycle-accurate simulator models execution time. For all systems, computation was scheduled to minimize energy, which led to the same schedule for all. To estimate power and area, the designs were synthesized with the Synopsis Design Compiler Synopsys for a TSMC 65nm library. The NBin and NBout SRAM buffers were modeled using CACTI Muralimanohar & Balasubramonian. The eDRAM area and energy were modeled with *Destiny* Poremba et al. (2015). To compare against *STR*, the per layer numerical representation requirements reported in Table 2 were found using the methodology of Judd et al. Judd et al. (2016b). All *PRA* configurations studied exploit software provided precisions as per Section 5.1. Section 6.4 analyzes the impact of this information on overall performance. All performance measurements are for the convolutional layers only which account for more than 92% of the overall execution time in *DaDN* Chen et al. (2014). *PRA* does not affect the execution time of the remaining layers.

6.1 Single- and 2-stage Shifting

This section evaluates the single-stage shifting *PRA* configuration of Sections 5– 5.1, and the 2-stage shifting variants of Section 5.1. Section 6.1 reports performance while Section 6.1 reports area and power. In this section, All *PRA* systems use pallet synchronization.

Performance: Figure 7 shows the performance of *STR* (leftmost bars) and of *PRA* variants relative to *DaDN*. The *PRA* systems are labelled with the number of bits used to operate the first-stage, weight shifters, e.g., the weight shifters of *"2-bit"* , or PRA_{2b}, are able to shift to four bit positions (0–3). "4-bit" or PRA_{4b}, is the single-stage *Pragmatic*, or PRA_{single} of Sections 5– 5.1 whose weight shifters can shift to 16 bit positions (0–15). It has no second stage shifter.

PRA_{single} improves performance by $2.59\times$ on average over *DaDN* compared to the $1.85\times$ average improvement with *STR*. Performance improvements over *DaDN* vary from $2.11\times$ for VGG19 to $2.97\times$ for VGGM. As expected the 2-stage *PRA* variants offer slightly lower performance than PRA_{single}, however, performance with PRA_{2b} and PRA_{3b} is always within 0.2% of PRA_{single}. Even PRA_{0b} which does not include any weight shifters outperforms *STR* by 20% on average. Given a set of oneffsets, PRA_{0b} will accommodate the minimum non-zero oneffset per cycle via its second level shifter.

Area and Power: Table 3 shows the absolute and relative to *DaDN* area and power. Two area measurements are reported: 1) for the unit excluding the SB, NBin and NBout memory blocks, and 2) for the whole chip comprising 16 units and all memory blocks. Since SB and NM dominate chip area, the per area area overheads Given the performance advantage of *PRA*, the area and power overheads are justified. PRA_{2b} is particularly appealing as its overall area cost over $BASE$ is only $1.35\times$ and its power $2.03\times$ while its performance is $2.59\times$ on average. Accordingly, we restrict attention to this configuration in the rest of this evaluation.

6.2 Per-column synchronization

Performance: Figure 8 reports the relative performance for PRA_{2b} with column synchronization and as a function of the number of SSRs as per Section 5.1. Configuration PRA_{2b}^{xR} refers to a

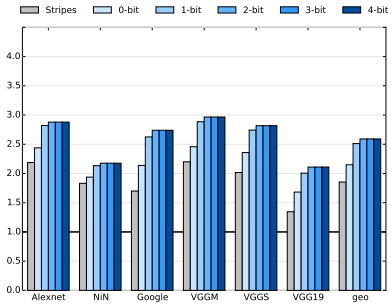


Figure 7: *Pragmatic*'s performance relative to DaDianNao using 2-stage shifting and per-pallet synchronization.

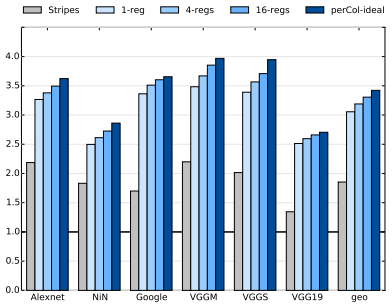


Figure 8: Relative performance of PRA_{2b} with column synchronization and as a function of the SB registers used.

	DDN	STR	0-bit	1-bit	2-bit	3-bit	4-bit
Area U.	1.55	3.05	3.11	3.16	3.54	4.41	5.75
Δ Area U.	1.00	1.97	2.01	2.04	2.29	2.85	3.71
Area T.	90	114	115	116	122	136	157
Δ Area T.	1.00	1.27	1.28	1.29	1.35	1.51	**1.75**
Power T.	18.8	30.2	31.4	34.5	38.2	43.8	51.6
Δ Power T.	1.00	1.60	1.67	1.83	2.03	2.33	**2.74**

Table 3: Area $[mm^2]$ and power $[W]$ for the unit and the whole chip. Pallet synchronization.

configuration using x SSRs. Even PRA_{2b}^{1R} boosts performance to $3.1\times$ on average close to the $3.45\times$ that is ideally possible with $PRA_{2b}^{\infty R}$.

Area and Power: Table 4 reports the area per unit, and the area and power per chip. The best performing PRA_{2b}^{1R} increases chip area by only $1.35\times$ and power by only $2.19\times$ over *DaDN*.

Energy Efficiency: Figure 10 shows the energy efficiency of various configurations of *Pragmatic*. *Energy Efficiency*, or simply *efficiency* for a system NEW relative to BASE is defined as the ratio $E_{\text{BASE}}/E_{\text{NEW}}$ of the energy required by BASE to compute all of the convolution layers over that of NEW. For the selected networks, *STR* is 16% more efficient than *DaDN*. The power overhead of PRA_{single} (PRA_{4b}) is more than the speedup resulting in a circuit that is 5% less efficient than *DaDN*. PRA_{2b} reduces that power overhead while maintaining performance yielding an efficiency of 28%. PRA_{2b}^{1R} yields the best efficiency at 48% over *DaDN*.

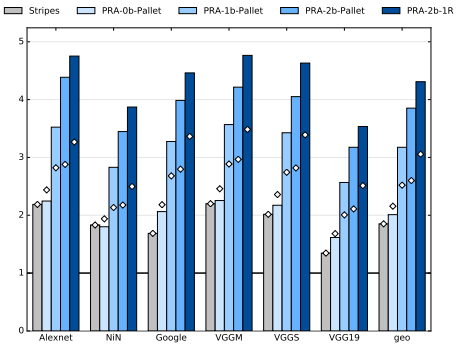


Figure 9: Relative performance of *Pragmatic* using Improved Oneffset Encoding for different configurations. Marked: performance not using IOE

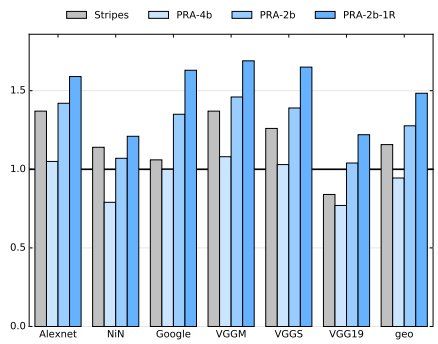


Figure 10: Relative energy efficiency

	DDN	STR	1-reg	4-reg	16-reg
Area U.	1.55	3.05	3.58	3.73	4.33
Δ Area U.	1.00	1.97	2.31	2.41	2.79
Area T.	90	114	122	125	134
Δ Area T.	1.00	1.27	1.36	1.39	**1.49**
Power T.	18.8	30.2	38.8	40.8	49.1
Δ Power T.	1.00	1.60	2.06	2.17	**2.61**

Table 4: Area $[mm^2]$ and power $[W]$ for the unit and the whole chip for column synchronization and PRA_{2b}.

Alexnet	**NiN**	**Google**	**VGGM**	**VGGS**	**VGG19**	**AVG**
23%	10%	18%	22%	21%	19%	19%

Table 5: Performance benefit due to software guidance

6.3 Improved Oneffset Encoding

Figure 9 reports performance for *Pragmatic* when using the enhanced oneffset generator described in Section 5.1. The considered configurations include PRA_{0b}, PRA_{1b} and PRA_{2b} (with pallet synchronization), and PRA_{2b}^{1R}. PRA_{0b} degrades by 7%, but the other configurations show improvements of 26%, 48%, and 41% respectively. A cause of degradation for PRA_{0b} is the increased spread of oneffset values (for example, the pair of neurons $011101, 010101$ takes 4 cycles with conventional encoding and 5 with enhanced encoding even though the total count of oneffsets is reduced from 7 to 6).

6.4 The Impact of Software

All *PRA* configurations studied thus far, used software provided per layer activation precisions to reduce essential bit content. *PRA* does not require these precisions to operate. Table 5 shows what fraction of the performance benefits is due to the software guidance for PRA_{2b}^{1R}, the best configuration studied. The results demonstrate that: 1) *PRA* would outperform the other architectures even without software guidance, and 2) on average, software guidance improves performance by 19%.

6.5 Quantization

Figure 11 reports performance for *DaDN* and *PRA* configurations using the 8-bit quantized representation used in Tensorflow Warden (2016); Google (2016). This quantization uses 8 bits to specify arbitrary minimum and maximum limits per layer for the activations and the weights separately, and maps the 256 available 8-bit values linearly into the resulting interval. This representation has higher

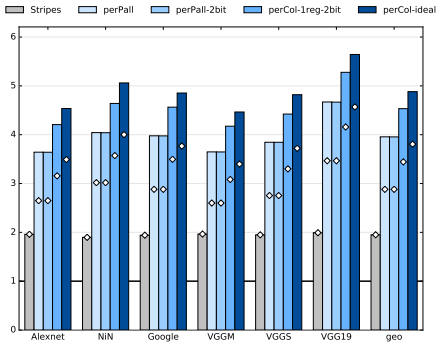


Figure 11: Performance: 8-bit quantized representation (marked: without IOE)

flexibility and better utilization than the reduced precision approach of *Stripes* since the range doesnt have to be symmetrical and the limits dont have to be powers of two, while still allowing straightforward multiplication of the values. The limit values are set to the maximum and the minimum activation values for each layer and the quantization uses the recommended rounding mode.

Figure 11 reports performance relative to *DaDN* for PRA_{single}, PRA_{2b}, PRA_{2b}^{1R}, and $PRA_{2b}^{\infty R}$. *PRA* performance benefits persist and are over $4.5\times$ for PRA_{2b}^{1R}. Measuring the area and energy of these designs is left for future work, however, the absolute area and energy needed by all will be lower due to the narrower representation. Moreover, given that the tile logic will occupy relatively less area for the whole chip and given that the SB and NM account for significant area and energy, the overall overheads of the *PRA* designs over *DaDN* will be lower than that measured for the 16-bit fixed-point configurations.

7 Related Work

The acceleration of Deep Learning is an active area of research and has yielded numerous proposals for hardware acceleration. *DaDianNao* (*DaDN*) is the de facto standard for high-performance DNN acceleration Chen et al. (2014). In the interest of space, this section restricts attention to methods that are either directly related to *DaDN*, or that follow a value-based approach to DNN acceleration, as *Pragmatic* falls under this category of accelerators. Value-based accelerators exploit the properties of the values being processed to further improve performance or energy beyond what is possible by exploiting computation structure alone. Cnvlutin Albericio et al. (2016) and *Stripes* Judd et al. (2016b)Judd et al. (2016a) are such accelerators and they have been already discussed and compared against in this work.

PuDianNao is a hardware accelerator that supports seven machine learning algorithms including DNNs Liu et al. (2015). *ShiDianNao* is a camera-integrated low power accelerator that exploits integration to reduce communication overheads and to further improve energy efficiency Du et al. (2015). Cambricon is the first instruction set architecture for Deep Learning Liu et al. (2016). Minerva is a highly automated software and hardware co-design approach targeting ultra low-voltage, highly-efficient DNN accelerators Reagen et al. (2016). Eyeriss is a low power, real-time DNN accelerator that exploits zero valued activations for memory compression and energy reduction Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne (2016). The Efficient Inference Engine (EIE) exploits efficient activation and weight representations and pruning to greatly reduce communication costs, to improve energy efficiency and to boost performance by avoiding certain ineffectual computations Han et al. (2016)Han et al. (2015). EIE targets fully-connected (FC) layers and was shown to be $12\times$ more efficient than *DaDN* on FC layers, and $2\times$ less efficient for convolutional layers. All aforementioned accelerators use bit-parallel units. While this work has demonstrated *Pragmatic* as a modification of *DaDN*, its computation units and potentially, its general approach could be compatible with all aforementioned accelerator designs. This investigation is interesting future work.

Profiling has been used to determine the precision requirements of a neural network for a hardwired implementation Kim et al. (2014). EoP has been exploited in general purpose hardware and other application domains. For example, Brooks *et al.* Brooks & Martonosi (1999) exploit the prefix bits due to EoP to turn off parts of the datapath improving energy. Park *et al.* Park et al. (2010), use a similar approach to trade off image quality for improved energy efficiency. Neither approach directly improves performance.

8 Conclusion

To the best of our knowledge *Pragmatic* is the first DNN accelerator that exploits not only the per layer precision requirements of CNNs but also the essential bit information content of the activation values. While this work targeted high-performance implementations, *Pragmatic*'s core approach should be applicable to other hardware accelerators. We have investigated *Pragmatic* only for inference and with image classification convolutional neural networks. While desirable, applying the same concept to other network types, layers other than the convolutional one, is left for future work. It would also be interesting to study how the *Pragmatic* concepts can be applied to more general purpose accelerators or even graphics processors.

REFERENCES

Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *2016 IEEE/ACM International Conference on Computer Architecture (ISCA)*, 2016.

A. D Booth. A signed binary multiplication technique. *The Quarterly Journal of Mechanics and Applied Mathematics*, 4(2):236–240, 1951.

David Brooks and Margaret Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, HPCA '99, pp. 13–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0004-8. URL http://dl.acm.org/citation.cfm?id=520549.822763.

Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, and O. Temam. Dadiannao: A machine-learning supercomputer. In *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 609–622, Dec 2014. doi: 10.1109/MICRO.2014.58.

Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pp. 262–263, 2016.

Zidong Du, R. Fasthuber, Tianshi Chen, P. Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and O. Temam. ShiDianNao: Shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, pp. 92–104, June 2015. doi: 10.1145/2749469.2750389. ShiDianNao.

Google. Low-precision matrix multiplication. https://github.com/google/gemmlowp, 2016.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. *arXiv:1510.00149 [cs]*, October 2015. URL http://arxiv.org/abs/1510.00149. arXiv: 1510.00149.

Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. *arXiv:1602.01528 [cs]*, February 2016. URL http://arxiv.org/abs/1602.01528. arXiv: 1602.01528.

Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, Raquel Urtasun, and Andreas Moshovos. Reduced-Precision Strategies for Bounded Memory in Deep Neural Nets, arXiv:1511.05236v4 [cs.LG] . *arXiv.org*, 2015.

Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor Aamodt, and Andreas Moshovos. Stripes: Bit-serial Deep Neural Network Computing . In *Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-49, 2016a.

Patrick Judd, Jorge Albericio, and Andreas Moshovos. Stripes: Bit-serial Deep Neural Network Computing . *Computer Architecture Letters*, 2016b.

Jonghong Kim, Kyuyeon Hwang, and Wonyong Sung. X1000 real-time phoneme recognition VLSI using feed-forward deep neural networks. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 7510–7514, May 2014. doi: 10.1109/ICASSP.2014.6855060.

Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. PuDianNao: A Polyvalent Machine Learning Accelerator. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pp. 369–381, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2835-7. doi: 10.1145/2694344.2694358. URL http://doi.acm.org/10.1145/2694344.2694358. PuDianNao.

Shaoli Liu, Zidong Du, Jinhua Tao, Dong Han, Tao Luo, Yuan Xie, Yunji Chen, and Tianshi Chen. Cambricon: An instruction set architecture for neural networks. In *2016 IEEE/ACM International Conference on Computer Architecture (ISCA)*, 2016.

Naveen Muralimanohar and Rajeev Balasubramonian. Cacti 6.0: A tool to understand large caches.

Jongsun Park, Jung Hwan Choi, and K. Roy. Dynamic Bit-Width Adaptation in DCT: An Approach to Trade Off Image Quality and Computation Energy. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 18(5):787–793, May 2010. ISSN 1063-8210. doi: 10.1109/TVLSI.2009.2016839.

M. Poremba, S. Mittal, Dong Li, J.S. Vetter, and Yuan Xie. Destiny: A tool for modeling emerging 3d nvm and edram caches. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2015*, pp. 1543–1546, March 2015.

Brandon Reagen, Paul Whatmough, Robert Adolf, Saketh Rama, Hyunkwang Lee, Sae Kyu Lee, Jos Miguel Hernndez-Lobato, Gu-Yeon Wei, and David Brooks. Minerva: Enabling low-power, highly-accurate deep neural network accelerators. In *International Symposium on Computer Architecture*, 2016.

Synopsys. Design Compiler. http://www.synopsys.com/Tools/Implementation/RTLSynthesis/DesignCompiler/Pages.

Christopher S. Wallace. A suggestion for a fast multiplier. *IEEE Trans. Electronic Computers*, 13 (1):14–17, 1964. doi: 10.1109/PGEC.1964.263830. URL `http://dx.doi.org/10.1109/PGEC.1964.263830`.

Peter Warden. Low-precision matrix multiplication. `https://petewarden.com`, 2016.

9 APPENDIX

9.1 *Pragmatic*'S POTENTIAL

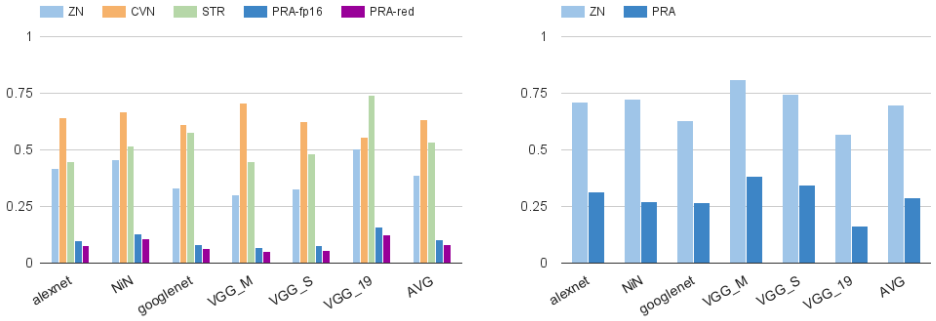
This appendix complements the analysis of Section 2 by estimating the potential of an idealized *Pragmatic* accelerator that can skip any term (product of a full precision weight and one input activation bit) while also improving execution time proportionally. Note the number of terms is considered *before* the Improved Oneffset Encoding described in Section 5.1 is applied.

To estimate *PRA*'s potential, this section compares the number of terms that would be processed by various computing engines for the convolutional layers of recent CNNs (see Section 6) for the two aforementioned baseline activation representations.

16-bit Fixed-Point Representation: The following computing engines are considered: 1) baseline representative of *DaDN* using 16-bit fixed-point bit-parallel units Chen et al. (2014), 2) a *hypothetical* enhanced baseline ZN, that can skip *all* zero valued activations, 3) Cnvlutin (CVN) a practical design that can skip zero value activations for all but the first layer Albericio et al. (2016), 4) *STR* that avoids EoP (see Table 2, Section 6) Judd et al. (2016b), 5) an ideal, software-transparent *PRA*, *PRA*-fp16 that processes only the essential activation bits, and 6) an ideal *PRA*, *PRA*-red, where software communicates in advance how many prefix and suffix bits can be zeroed out after each layer (see Section 5.1).

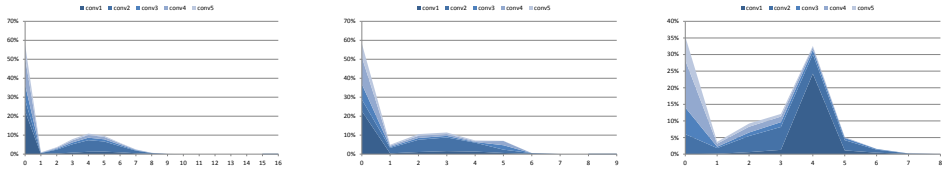
Figure 12a reports the number of terms normalized over *DaDN* where each multiplication is accounted for using an equivalent number of terms or equivalently additions: 16 for *DaDN*, ZN, and CVN, p for a layer using a precision of p bits for *STR*, and the number of essential activation bits for *PRA*-fp16, and for *PRA*-red. For example, for $n = 10.001_{(2)}$, the number of additions counted would be 16 for *DaDN* and CVN+, 5 for *STR* as it could use a 5-bit fixed-point representation, and 2 for *PRA*-fp16 and *PRA*-red.

On average, *STR* reduces the number of terms to 53% compared to *DaDN* while skipping just the zero valued activations could reduce them to 39% if ZN was practical and to 63% in practice with CVN. *PRA*-fp16 can ideally reduce the number of additions to just 10% on average, while with software provided precisions per layer, *PRA*-red reduces the number of additions further to 8% on



(a) 16-bit fixed-point (b) 8-Bit Quantized

Figure 12: Convolutional layer computational demands



(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized

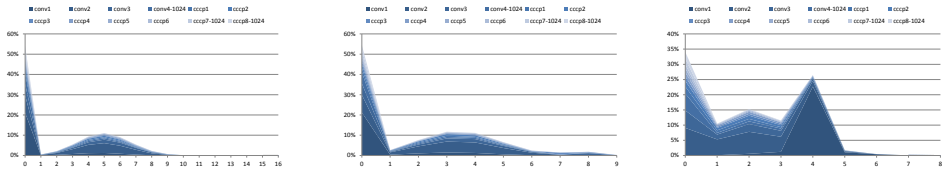
Figure 13: AlexNet: Per Layer '1'-bit Count Distributions.

average. The potential savings are robust across all CNNs remaining above 87% for all DNNs with *PRA*-red.

8-bit Quantized Representation: Figure 12b shows the relative number of terms processed for: 1) a bit-parallel baseline, 2) an ideal, yet impractical bit-parallel engine that skips all zero activations, and 3) *PRA*. In the interest of space and since *PRA* subsumes *STR* and CVN they are not considered. *Pragmatic*'s benefits are significant even with an 8-bit quantized representation. On average, skipping all the zero valued activations would eliminate only 30% of the terms whereas *Pragmatic* would remove up to 71% of the terms.

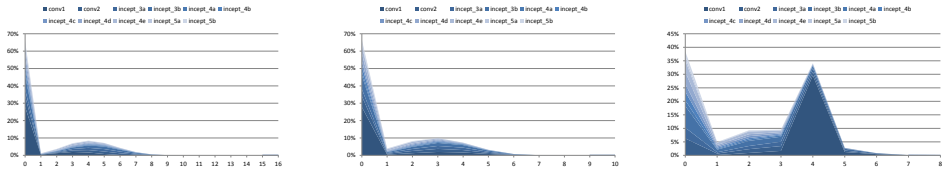
9.2 Essential Bit Content Distributions

This section reports the distributions of the essential bit count for the activations processed per convolutional layers for the networks studied. Three distributions are shown per network for the activations for three different representations: 1) 16-bit fixed-point, 2) per layer fixed-point, and 3) 8-bit Quantized. A peak appears for values having four bits that are 1 for the quantized representation since the value zero is mapped to a non-zero index having four bits that are one (114). Note that, as in Section 9.1, the distributions are taken before Improved Oneffset Encoding.



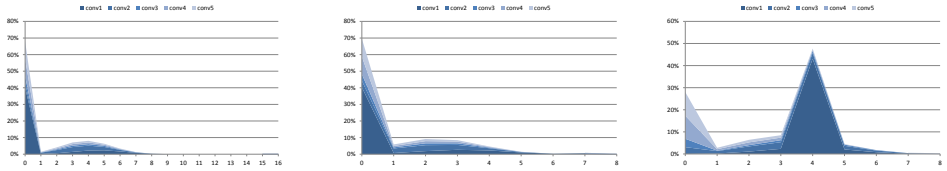
(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized: Activations

Figure 14: NiN: Per Layer '1'-bit Count Distributions.



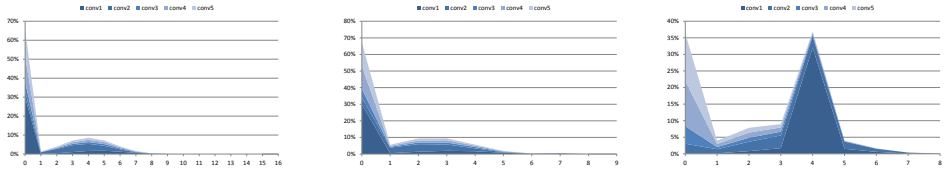
(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized: Activations

Figure 15: GoogLeNet: Per Layer '1'-bit Count Distributions.



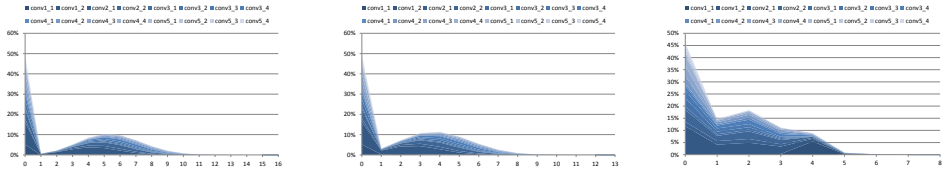
(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized: Activations

Figure 16: VGG_M: Per Layer '1'-bit Count Distributions.



(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized: Activations

Figure 17: VGG_S: Per Layer '1'-bit Count Distributions.



(a) 16-bit: Full-Precision (b) 16-bit: Per Layer Precision (c) Quantized: Activations

Figure 18: VGG_19: Per Layer '1'-bit Count Distributions.