BIT-PRAGMATIC DEEP NEURAL NETWORK COMPUTING

Jorge Albericio[1], Patrick Judd[2], Alberto Delmas[2], Sayeh Sharify[2] & Andreas Moshovos[2]
[1]NVIDIA* [2]University of Toronto
jalbericiola@nvidia.com
{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

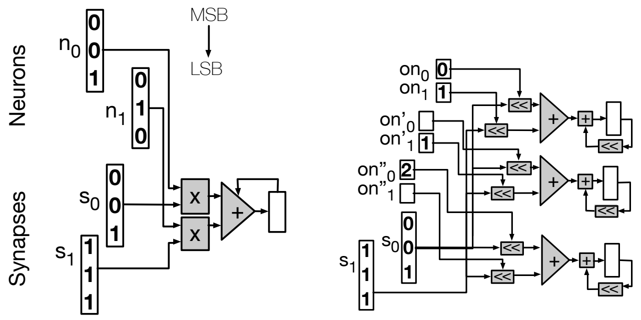
In image classification using Convolutional Neural Networks (CNNs), convolutional layers account for most of the execution time. Deep learning hardware typically uses either 16-bit fixed-point (Chen et al. (2014)) or quantized 8-bit numbers (Warden (2016); Google (2016)) and bit-parallel compute units which perform many ineffectual computations. Specifically, convolutional layers perform 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 participates in a multiplication it adds nothing to the output activations. This work shows how these ineffectual multiplications can be avoided improving overall performance and energy by targeting, as a first step, the ineffectual bits of activations only.

This work presents *Pragmatic* (*PRA*) a DNN accelerator whose goal is to process only the *essential* (non-zero) bits of the input activations. Conceptually, the idea behind *PRA* would appear deceptively simple: process the activations bit-serially compensating for the loss in computation bandwidth by exploiting the abundant parallelism of convolutional layers. Processing activations bit-serially enables *PRA* to skip the zero bits. However, a straightforward implementation of a bit-serial processing engine proves impractical as it suffers from unacceptable energy and area overheads. To improve performance and energy efficiency over a state-of-the-art bit-parallel accelerator and without a disproportionate increase in area, *PRA* employs the following five 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 *Stripes* of Judd et al. (2016b;a) (*STR*) but adapted for the aforementioned representation, 3) judicious SIMD (single instruction multiple data) lane grouping to

*This work was completed while Jorge Albericio was a Postdoctoral Fellow at the University of Toronto

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

maintain wide memory accesses and to avoid fragmenting and enlarging the on-chip weight memories, 4) computation re-arrangement to reduce datapath area, and 5) Booth encoding to further reduce the number of essential bits per activation.

For a Machine Learning practitioner *PRA* introduces an additional dimension, that of controlling the essential bit content of activations, upon which they can improve performance and energy efficiency. This work explores such an alternative, where the software explicitly communicates how many prefix and suffix bits to discard after each layer.

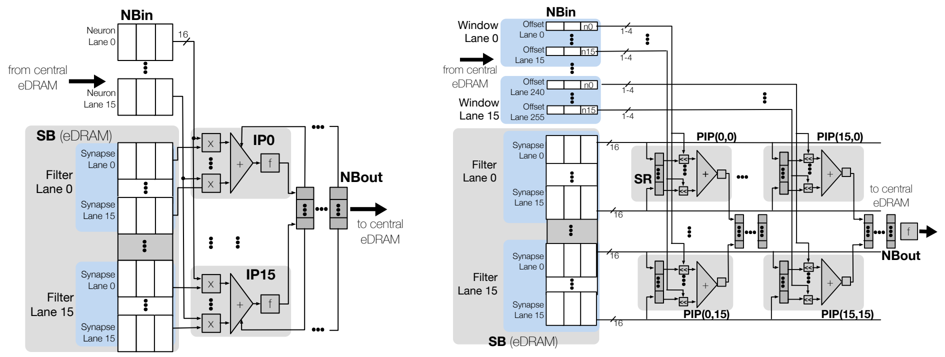
2 Potential

Table 1 shows that in recent image classification networks on average 93% and 69% of activation bits are zero and thus ineffectual during multiplication when using respectively 16-bit fixed-point and 8-bit quantized representations (Warden (2016); Google (2016)). The table also shows that the bias toward zero bits remains strong even when considering only the non-zero activations (NZ rows).

3 *Pragmatic*: A Simplified Example

This section illustrates the idea behind *Pragmatic* via a simplified example. The bit-parallel unit of Figure 1a multiplies two activations with their respective weights and via an adder reduces the two products. This unit processes four ineffectual terms as two sources of inefficiency, *excess of precision* (EoP) and *lack of explicitness* (LoE) manifest here: n_0 and n_1 are represented using 3 bits instead of 2 respectively due to EoP and even in 2 bits, they each contain a zero bit due to LoE. In general, this unit will take $\lceil N/2 \rceil$ cycles to process N activation and weight pairs, regardless of their precision and the essential bit content of the activations.

In the simplified *PRA* engine of Figure 1b activations are represented as vectors of essential bit offsets, or *oneffsets* and are processed "bit-serially". For example, activation $n_0 = 001_{(2)}$ is represented



(a) (b)

Figure 2: a) DaDianNao Tile. b) Pragmatic Tile.

as $on_0 = (0)$, and an activation value of $111_{(2)}$ would be represented as $(2, 1, 0)$. An out-of-band bit not shown indicates the activation's end. A shifter per weight multiplies the weight with the corresponding activation oneffset prior 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 1a, *PRA* takes advantage of weight reuse and processes multiple activations groups in parallel. 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.

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.

4 *Pragmatic*

Pragmatic is demonstrated as a modification of the *DaDianNao* accelerator (*DaDN*) proposed by Chen et al. (2014) for ease of comparison, however, the architecture is configurable. Figure 2a shows a *DaDN* tile processes 16 filters concurrently calculating 16 activation and weight products per filter for a total of 256 products per cycle[1]. A *DaDN* chip contains 16 tiles and weights and activations use a 16-bit fixed-point representation.

Input Activation Representation: *PRA* converts each activation into an 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 the 2MB *Neuron Memory* (NM), and converted on-the-fly in the *PRA* representation as they are broadcast to the tiles.

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. To guarantee that *PRA* always performs as well as *DaDN* it should process $4K$ terms per cycle. 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 each from a different window and which are processed bit-serially.

[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.

For example, in a single cycle a *PRA* title processing filters 0 through 15 could combine combine $s_B^0(x,y,0), ..., s_B^1 5(x,y,0)$ 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 $on(x+15,y,4)$. 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: Since the number of oneffsets will vary per activation, each neuron/activation 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 impractical as it would require partitioning and replicating the weight memory (SB) so that 4K unrelated weights could be read per cycle, and it would also increase activation memory (NM) complexity and bandwidth.

PRA avoids these complexities 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. Finer synchronization schemes are possible taking advantage of the fact that the weights are not being read every cycle. We found that column synchronization, where each group of 16 neuron lanes can advance independently offers a good performance, area overhead, and complexity compromise.

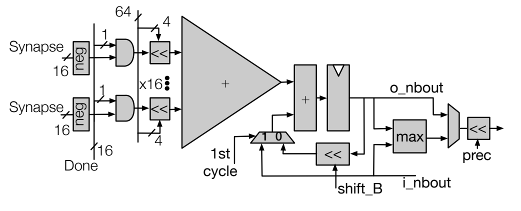


Figure 3: Pragmatic Inner Product Unit.

Pragmatic Tile: Figure 2b 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.

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. 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. Figure 3 shows the resulting PIP.

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 $a+1, -b$. *Pragmatic* uses a modified Booth encoding that 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.

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 judi-

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.

cious 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. For example, using the profiling method of Judd *et al.,* Judd et al. (2015), software can communicate the precisions needed by each layer as meta-data. The hardware trims the output activation precision on-the-fly.

5 Evaluation Summary

The performance, area and energy efficiency of *Pragmatic* is compared against *DaDN* Chen et al. (2014) the fastest bit-parallel accelerator proposed to date that processes all activations regardless of their values.

Methodology: The same methodology is consistently used for all systems. 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 and their layout was produced 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)). The per layer numerical representation requirements reported in Table 2 were found using the methodology of Judd et al. (2016b) and are used to trim activations prior to oneffset generation in *PRA*. All *PRA* configurations studied exploit software provided precisions. 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.

Performance: Due to space limitations we omit the design exploration and limit our attention to a *PRA* configuration that uses 2-stage shifting with 4-way first-level shifters, column-synchronization and one extra register to reduce weight memory accesses. Figure 4 reports the performance improvement with *PRA* over an equivalent *DaDN* configuration for the conventional 16-bit fixed-point representation (16b) and for a Tensorflow-like (Warden (2016); Google (2016)) 8-bit quantized representation (8bQ). On average, *PRA* is 3.0x and 3.4x faster than *DaDN* for the two representations respectively. When the improved oneffset encoding is used, on average *PRA* is 4.3x (IE-16b) and 4.5x (IE-8bQ) faster than *DaDN* for the two representations respectively.

Area, Power, Energy: Table 3 reports the area and power of *DaDN* and *PRA* for the 16-bit fixed-point configuration only. *PRA* units are 2.31x larger, but overall the area overhead is only 1.36x for the whole chip. Similarly, the power dissipated by the units with *PRA* is 2.41x compared to *DaDN* and 2.06x for the whole chip. However, as the next section shows, as *Pragmatic* is considerably faster, it is more energy efficient even when operating at the same frequency and voltage as *DaDN*. If further efficiency is required, frequency and voltage scaling can be used while maintaining a performance advantage over *DaDN*.

Energy Efficiency:

Figure 4 reports the relative energy efficiency of *PRA* over that of *DaDN* for the 16-bit fixed-point representation (EE-16b) with zero-skipping. On average *PRA* is 1.44x more energy efficient when

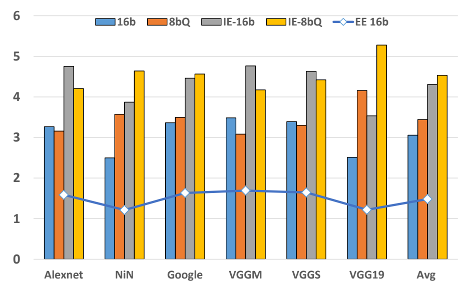


Figure 4: Performance and Energy Efficiency improvement with *PRA* over *DaDN*.

	DDN	*PRA*
Area Unit	1.55	3.58
Δ Area Unit	1.00	2.31
Area Chip	90	122
Δ Area Chip	1.00	1.36
Power Chip	18.8	38.8
Δ Power Chip	1.00	2.06

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

operating at the same voltage and frequency. As *PRA* is faster, voltage and frequency scaling can be used to improve energy efficiency further while maintaining a performance advantage over *DaDN*. Modeling the energy efficiency of the improved oneffset generators and the improved encoding configuration is left for future work.

6 Related Work

The acceleration of Deep Learning is an active area of research and has yielded numerous proposals for hardware acceleration. *PRA* is closely related to *Stripes* which uses a bit-serial/bit-parallel approach to convert reduced precision requirements to performance (Judd et al. (2016a)). *PRA* would appear to be a straightforward extension of *STR* that simply skips zero activation bits. However, a straightforward modification of *STR* is impractical as it yields unacceptable area and energy overheads: 1) Since each cycle each weight is multiplied by a different power of two, twice as wide reduction adders would be needed to accommodate the case where one weight is multiplied by 2^0 and another with 2^{max} (circuit-level synthesis shows that the execution unit area would increase to 3.7x compared to the bit-parallel *DaDN*). 2) Since the essential bit content will vary per activation, the activation accesses would no longer be synchronized. Instead of one wide access for all activations, the activation memory would have to somehow to support multiple single activation requests. 3) For the same reason as in 2), memory accesses for weights will increase as different activations will need to be accessing a different set of weights. Instead of a single wide access once for each set of activations, the weight memory will have to support one wide access per individual activation. For the configuration studied, the weight memory would have to provide up to 256 different sets of 256 weights on average every 4 cycles.

Appendix 7 reviews additional relevant work.

7 Conclusion

To the best of our knowledge *Pragmatic* is the first CNN 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 CNNs. 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.

ACKNOWLEDGEMENTS

This work was supported by an Natural Sciences and Engineering Research Council of Canada Discovery Grant and Accelerator Supplement.

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.

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.

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.

APPENDIX

This section reviews additional work related to *PRA*. In the interest of space, this section restricts attention to methods 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) is a wide-SIMD accelerator that skips computations involving ineffectual activations (either having the value zero or being close to zero). *PRA* has a similar effect as it would only process the essential bit content. 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.

Kim et al. (2014) used profiling has been used to determine the precision requirements of a neural network for a hardwired implementation. EoP has been exploited in general purpose hardware and other application domains. For example, 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.