Moving CNN Accelerator Computations Closer to Data

Sumanth Gudaparthi
University of Utah
Email: sgudapar@cs.utah.edu

Surya Narayanan
University of Utah
Email: surya@cs.utah.edu

Rajeev Balasubramonian
University of Utah
Email: rajeev@cs.utah.edu

***Abstract*—A significant fraction of energy in recent CNN accelerators is dissipated in moving operands between storage and compute units. In this work, we re-purpose the CPU's last level cache to perform in-situ dot-product computations, thus significantly reducing data movement. Since a last level cache has several subarrays, many such dot-products can be performed in parallel, thus boosting throughput as well. The in-situ operation does not require analog circuits; it is performed with a bit-wise AND of two subarray rows, followed by digital aggregation of partial sums. The proposed architecture yields a 2.74× improvement in throughput and a 6.31× improvement in energy, relative to a DaDianNao baseline. This is primarily because the proposed architecture eliminates a large fraction of data transfers over H-Tree interconnects in the cache.**

***Keywords*-CNN, neuromorphic architectures, neural networks, accelerator**

I. Introduction

Several recent works have introduced accelerators for convolutional neural networks (CNNs) [1], [2], [3], [4], [5], [6]. Many of these papers achieve high efficiency by reducing data movement and with new dataflows that maximize data reuse. Some even propose in-situ computing to reduce data movement. However, prior efforts at in-situ computing either require inefficient analog circuits (ISAAC [2], PRIME [7]), or require a new higher-cost DRAM chip (DRISA [8]). This paper introduces an *SRAM cache based In-Situ Computation Accelerator (SISCA)*, that performs dot products adjacent to SRAM subarrays. This significantly reduces data movement, it performs in-situ computations without relying on analog circuits, and it shows how the last level cache (LLC) in a conventional CPU can be re-purposed as a CNN accelerator. This can improve CNN inference efficiency in both servers and mobile devices.

SISCA leverages recently introduced logic-in-memory circuits [9] that activate two wordlines in an SRAM subarray to produce the bit-wise AND of the contents in the corresponding rows. A tree of adders beside the SRAM subarray then adds these bits to produce the eventual dot-product. To facilitate these operations, one of the operands in the dot-product is shifted and replicated in the SRAM.

Several factors contribute to the high efficiency of this architecture. It offers very high parallelism since every subarray in a large LLC can perform parallel dot-product operations. Weights once programmed into a subarray do not have to navigate the H-tree within the LLC bank. The primary downside is that the accelerator offers a lower on-chip storage capacity than some competing accelerators. This is because SISCA relies on SRAM (as opposed to dense memristors in ISAAC/PRIME or eDRAM in DaDianNao [1]) and because of operand replication. However, its integration into an LLC means that the incremental cost of adding a CNN accelerator to a mobile or server processor is relatively small.

Our preliminary evaluation with ResNet18 shows that SISCA is able to achieve 2.74× higher throughput than DaDianNao and 6.31× lower energy.

II. Background

A. Logic-in-Memory

Work by Jeloka et al. [9] introduces a configurable SRAM memory that can perform bit-wise logical operations on two or more rows within a subarray. By activating multiple wordlines, logical AND/NOR operations can be performed on cells that share the same bitlines. This Logic-in-Memory operation splits the cross-couple of a conventional voltage differential sense amplifier (SA) into two parallel cross-couples tied together.

B. Compute Caches

The Compute Cache architecture [10] employs the Logic-in-Memory circuit to perform several simple in-situ vector operations (copy, search, compare, and logical operations). With in-situ computing, the on-chip data movement and especially traversal of the H-Tree are reduced. Since the H-Tree consumes more than 80% of energy in a large cache, this has a significant impact on overall cache energy and throughput.

C. CNN Accelerators

Our baseline for this study is DaDianNao [1], which offers the basic scaffolding required in most CNN accelerators. For future work, we will explore if the proposed SISCA architecture can be combined with the many ideas that have been introduced in recent years, e.g., the data reuse in Eyeriss [3], or the deep compression in EIE [5].

A few other CNN accelerators, notably ISAAC [2] and PRIME [7], have constructed pipelines that incorporate in-situ dot-product calculations. Memristor crossbars are used to compute an analog dot-product across several rows and

978-1-5386-7367-6/18/$31.00 ©2018 IEEE
DOI 10.1109/EMC2.2018.00015

in parallel for several columns. That approach requires large and power-hungry analog-to-digital converters. Analog circuits have traditionally also been unattractive for industry. This paper uses a more modest digital in-situ operation that may be more palatable to industry. We therefore arrive at a design point that offers performance and energy that is between those of DaDianNao and ISAAC.

III. Proposal

A. Overview

We propose incorporating the Logic-in-Memory operation into a processor's LLC. This introduces a small overhead in area, but has a minimal impact on the typical read/write operations of the LLC. When executing a CNN, the LLC or parts of it can be operated as an accelerator. To process a layer of the CNN, its input feature maps and weights are loaded from memory into LLC subarrays. In-situ computations are performed and the output results are retained in the LLC subarrays. When performing the next layer, a new set of weights are loaded into LLC subarrays and the computations continue. Many networks may be able to accommodate all their weights in the LLC, thus avoiding frequent memory fetches.

The multiplication of two operands can be computed by AND-ing each bit of one operand with each bit of the other operand, and aggregating the results with appropriate shifts. To facilitate the AND operation, we leverage the Logic-in-Memory circuit; to reduce data movement, the aggregation is performed adjacent to the subarray without H-Tree traversal.

The weights and the input feature maps are first placed into SRAM subarrays. If the weights have a smaller size than the input feature maps, the weights are replicated. Each replica occupies a different row in the subarray and is a bit-shifted version of the previous row. A row of the input feature map is bit-wise AND-ed with a row of weights; this is repeated for each shifted replica. Once this is done (i.e., each bit of an input operand has been AND-ed with every bit of a weight operand), an adjacent *RAT unit* (Registers and Adder Tree) aggregates all these partial sums to yield a product.

To compute an output neuron, a number of such products have to be aggregated. Since a subarray row can have many operands, the above steps yield many products within the subarray. These products can be aggregated in the RAT unit adjacent to the subarray (without requiring H-Tree traversal). Another aggregation step must be performed if a neuron has many inputs and its dot-product computation is spread across multiple subarrays.

To compute other output neurons in a convolutional layer, the input operands in a row can be shifted so they can be multiplied with other weights in that subarray. Further, a row of input operands must also be sent to other subarrays so they can be multiplied with weights in other subarrays. Thus, to compute all the output neurons in a convolutional layer, the input operand rows must be occasionally moved around over the H-Tree. However, this data movement is much lower than the data movement required in a baseline like DaDianNao.

Since frequent bit-shift and operand-shift operations must be performed for activations and input operands respectively, each bank has a Shifter unit that is shared by all the subarrays. For example, when loading weights into the subarrays, the Shifter unit receives a row of weights from off-chip DRAM; shifted replicas of that row are then written into the subarrays.

B. Example

Figure 1 shows the overall block diagram of the SISCA architecture and an example mapping of operands to one subarray. The figure shows a single LLC bank that has 8 subarrays (only 4 are shown) and a shifter unit. Each subarray has a RAT unit. We assume that each subarray size is 512×512. We split this subarray into 32 logical columns; each logical column is wide enough to store a 16-bit operand.

Consider an example layer in a CNN where 128 $\{3\times3\times64\}$ kernels are being applied on $128\times128\times64$ input feature maps. Assume that inputs and weights are both 16-bit fixed-point values.

There are a total of 1 million activations and 72K weights present in this example. As the set of weights is significantly smaller than the set of activations, the weights are replicated 16× and are placed in different rows of a subarray. Each replica in a logical column is a 1-bit left rotated version of the previous replica (see Figure 1b). For example, a weight operand[1] (W_0) is replicated 16× and its left shifted versions are placed in different rows (*row:1* to *row:16*) of the same logical column (LC:1).

The rows in a subarray are split between weights and activations. To utilize the computational capability of SISCA, the activations and kernels are distributed evenly across all 1024 subarrays. For our example, the activations have a memory footprint of 2 MB, while the replicated kernels have a memory footprint of 2.25 MB. After even distribution among all the subarrays, each subarray has 32 rows allocated for activations, and 36 rows allocated for kernels. The remaining 444 rows are used to store the generated output neurons as well as to store the weights for other layers.

Figure 1b shows the operations that are performed in a single subarray. The kernel entry in *row:1* is bit-wise AND'ed with the feature map entry in *row:p+1*. This operation generates $1/16^{th}$ of the bits required for one multiplication operation. The partial products after an AND operation are stored in registers present in the RAT unit. To derive the next set of bits required for the multiplication operation, a bit-wise AND is performed for the kernel entry

[1] C_{a-b}: bit number-b in the a^{th} variable of operand C

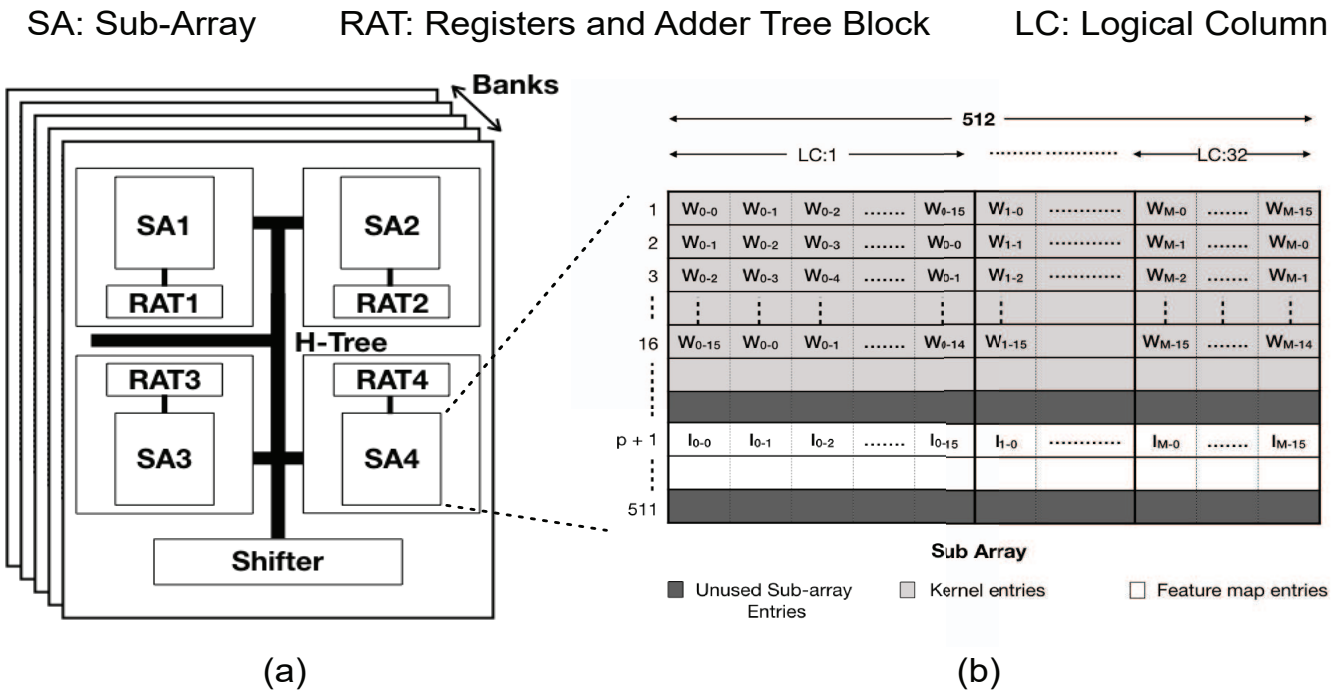

Figure 1. SISCA Block Diagram

in *row:2* and the the feature map entry in *row:p+1*. After 16 such steps until *row:16*, all the partial products necessary for the multiplication operation have been generated and stored in registers. These partial products are sent to a tree of 1-bit adders present in the RAT unit for accumulation. A single output neuron in this example must sum 576 products. Therefore, this computation is spread across 18 subarrays (each subarray performs 32 products in parallel). A home subarray gathers these partial sums to produce the final output neuron; 34 bytes are transmitted on the H-tree every 16 cycles to enable this. Once the output neuron is produced, it is written to one of the rows in the home subarray. Since a subarray has two rows of weights (and their 30 rotated replicas), the same row of input operands can be reused to produce another set of output neurons in the next 16 cycles.

In the next step, each subarray uses the next row of inputs to compute the next two output neurons in the next 32 cycles. Since each subarray in our example has 32 rows of inputs, this process continues for a total of 1K cycles, generating 64 output neurons across 18 subarrays. Since the LLC has 1K subarrays, nearly 3.6K output neurons are generated in these 1K cycles with minimal data movement.

Next, each input row in each subarray must perform an operand-level rotation, i.e., input operand I_0 occupies the position of input operand I_1, and so on. This shift is performed by sending each input operand row to its shifter over the H-tree. One such shift has to be performed every 32 cycles, so the overhead is low. After these shifts, another 3.6K output neurons are generated in the next 1K cycles. Since a row has 32 input operands, this process repeats for a total of 32K cycles, generating 115K output neurons.

Once this is done, every row of inputs has to move to a different subarray so it can be multiplied with other weights. This H-tree traversal for a row is performed once every 1K cycles, so the overhead is again low.

To summarize, processing a layer requires many hierarchical steps. First, it takes 16 cycles to multiply two 16-bit operands. Several such products are computed in parallel. These products are aggregated, partially within a subarray, and partially across subarrays to produce a set of output neurons. The second hierarchical step is to repeat the above computation for all rows of input feature maps and kernel weights in a subarray. The next hierarchical step is to rotate the input operands in a row and repeat. The final hierarchical step is to send input operand rows to other subarrays so additional convolutions can be performed.

IV. Methodology

We modelled the energy and area numbers of the cache architecture and registers using CACTI 6.5 [11] at 32nm

technology. The area and energy numbers for the adder tree are derived from [12], while the numbers for the barrel right rotate only shifter circuit is based on [13]. The energy overhead of activating multiple wordlines in a subarray, and the area overhead in modifying the sense-amplifier circuit are adapted from the analysis of Compute Caches [10]. We used a 32 MB SRAM for most of our analysis. The additional hardware enhancements to the LLC constitute an area overhead of 18% for the LLC. The area and energy estimates for DaDianNao are taken directly from that work [1], but scaled from 28 nm to 32 nm for an apples-to-apples comparison.

We have manually mapped a version of the ResNet18 deep network [14] to individual subarrays using the mapping technique described in Section III-B. We made sure no hazards or conflicts occur as the pipeline progresses. We also mapped the network manually on DaDianNao.

V. Results

We first analyze the impact of the proposed accelerator on performance. Figure 2 depicts the execution time for each layer of ResNet18 on SISCA and DaDianNao. We observe that SISCA outperforms DaDianNao in all the convolution layers (C1-17). The overall execution time reduces by 2.74×. The main reason for this is the high parallelism offered by SISCA's in-situ computations (32K computational units in a 32MB SRAM with 16-bit operands). However, DaDianNao has lower execution time for the fully connected layers (FC). This is because the FC layer in ResNet18 only computes 1K activations from 1×1×512 input feature map. To better utilize all the computational units in SISCA, we duplicate these 512 feature maps to cover all the subarrays, thus increasing the utilization of the H-Tree. In this workload, the FC layer plays a small role in overall execution time.

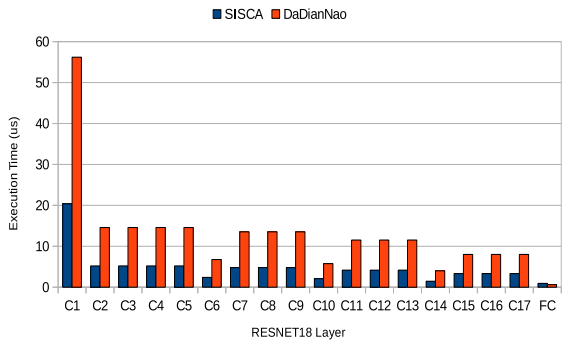

Figure 2. Execution time comparison for each layer of ResNet18 for SISCA and DaDianNao.

We define Computational efficiency (CE) as the number of 16-bit operations performed per second per mm^2 (GOPS/s x mm^2). Executing the ResNet18 network over both architectures gives us a CE of 18.6 GOPS/s x mm^2 for DaDianNao, and 26.8 GOPS/s x mm^2 for SISCA. While SISCA can offer higher throughput than DaDianNao, its area overhead is non-trivial. This is primarily because DaDianNao uses eDRAM ($88mm^2$ for 36 MB), while SISCA uses SRAM ($167mm^2$ for 32 MB). Viewed through a different lens, SISCA allows an LLC to be re-purposed as a CNN accelerator with an additional $25mm^2$ area overhead (18%) w.r.t. baseline cache.

SISCA energy breakdown per subarray for a block size of 64 Bytes	
Read Energy	147.4 pJ
H-Tree Energy	146.3 pJ
Subarray Access Energy	1.039 pJ
Energy per bit-wise AND Op across two wordlines	2.60 pJ
Energy of RAT block	8.64 pJ
DaDianNao energy breakdown for 8KB block sizes	
NFU (spread across 16 tiles) Energy	3.8 nJ
Central eDRAM Energy (4MB)	3.73 nJ
Tile eDRAM Energy (32 MB)	8.96 nJ

Table I
SISCA Energy Breakdown.

As most of the computations are in-situ, the H-Tree or the output drivers of the SRAM are used minimally for neural network computations. Table I shows the energy breakdown for each SISCA and DaDianNao operation. Note that the subarray read and H-Tree energy are infrequently exercised in SISCA. The energy for subarray access, bit-wise AND, and RAT units are significantly lower for SISCA, even if they are multiplied by 1K (since there are 1K subarrays).

Figure 3 compares the energy consumption for DaDianNao and SISCA for each layer of ResNet18. SISCA yields a 6.31× energy reduction primarily because of much fewer H-tree traversals.

VI. Conclusion

In this paper, we propose that with some hardware enhancements, the SRAM cache can function as a neural network accelerator. This can be helpful for both server architectures and mobile devices. Our early estimates show that relative to DaDianNao, SISCA offers high throughput and energy efficiency. This is primarily because of the massive data parallelism that can be achieved by activating multiple wordlines among all the subarrays, and the reduced data movement on H-Tree interconnects. While prior works like DaDianNao have leveraged a near-data processing approach, we show that this concept can be further exploited by moving computations even closer to data storage in subarrays.

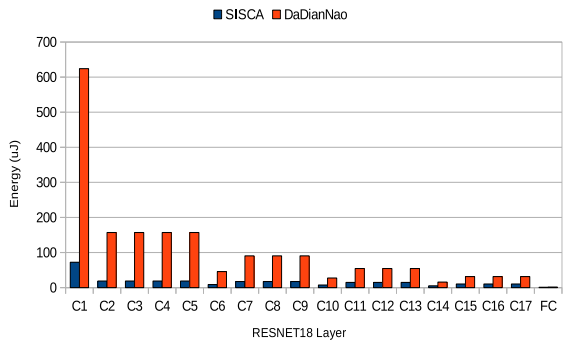

Figure 3. Energy consumption for each layer for SISCA and DaDianNao for ResNet18.

REFERENCES

[1] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun *et al.*, "DaDianNao: A Machine-Learning Supercomputer," in *Proceedings of MICRO-47*, 2014.

[2] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. Strachan, M. Hu, R. Williams, and V. Srikumar, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," in *Proceedings of ISCA*, 2016.

[3] S.Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. Horowitz, and W. Dally, "EIE: Efficient Inference Engine on Compressed Deep Neural Network," in *Proceedings of ISCA*, 2016.

[4] S.Han, H. Mao, and W. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization, and Huffman Coding," in *Proceedings of ICLR*, 2016.

[5] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks," in *Proceedings of ISCA-43*, 2016.

[6] S. Venkataramani, A. Ranjan, S. Avancha, A. Jagannathan, A. Raghunathan, S. Banerjee, D. Das, A. Durg, D. Nagaraj, B. Kaul, and P. Dubey, "SCALEDEEP: A Scalable Compute Architecture for Learning and Evaluating Deep Networks," 2017.

[7] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A Novel Processing-In-Memory Architecture for Neural Network Computation in ReRAM-based Main Memory," in *Proceedings of ISCA-43*, 2016.

[8] S. Li, D. Niu, K. T. Malladi, H. Zheng, B. Brennan, and Y. Xie, "Drisa: A dram-based reconfigurable in-situ accelerator," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, 2017.

[9] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, "A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.

[10] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 481–492.

[11] N. Muralimanohar *et al.*, "CACTI 6.0: A Tool to Understand Large Caches," University of Utah, Tech. Rep., 2007.

[12] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*. IEEE, 2014, pp. 10–14.

[13] S. Huntzicker, M. Dayringer, J. Soprano, A. Weerasinghe, D. M. Harris, and D. Patil, "Energy-delay tradeoffs in 32-bit static shifter designs," in *2008 IEEE International Conference on Computer Design*, 2008, pp. 626–632.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv preprint arXiv:1512.03385*, 2015.