SISCA: An SRAM In-Situ Computataion Accelerator

Sumanth Gudaparthi
Department of
Computer Science
Email: sgudapar@cs.utah.edu

Krithika Ganesh Sundaram Iyer
Department of
Electrical and Computer Engineering
Email: krithikags.iyer@utah.edu

***Abstract*—Various number of machine learning accelerators have been proposed in the recent years. Data movement between memory and computational units in conventional architectures is creating a memory wall bottleneck. To address this challenge, we propose SISCA, an SRAM in-situ computational accelerator to provide massive compute capability with high memory bandwidth. Our first contribution is modifying the SRAM architecture to accomodate in-situ multiplication operations. Our second contribution is to design a hierarchical memory architecture with SRAM and eDRAM to address the memory limitation caused due to the area overhead of SRAM. On running over AlexNet and VGG workloads, our architecture given an improvement of 12x and 14x on throughput and energy with an area overhead of 2.5x relative to DaDianNao.**

I. Introduction

With the advent of machine learning we are witnessing a wide variety of loads being processed, which require faster and efficient processing. For many of these applications, local processing near the sensor is preferred over the cloud due to privacy or latency concerns, or limitations in the communication bandwidth and accelerators are the solutions for this.

In the recent times a wide variety of accelerator chips are being proposed for the tasks of machine learning. In these accelerators, it is important to balance the accuracy, energy, throughput and cost requirements. Since data movement dominates energy consumption, the primary focus of recent research has been to reduce the data movement while maintaining accuracy, throughput and cost. This means selecting architectures with favourable memory hierarchies. Accelerators provide an opportunity to optimize the data movement (i.e., dataflow) to minimize accesses from the expensive levels of the memory hierarchy.

Considerable work has been done to bridge the gap between faster computations and memory bottleneck issue. Few of the approaches are, to build a computation intensive architecture while using a huge amount of memory on-chip, or to build a compute capable memory using a processing-in-memory architecture.

Emerging machine learning algorithms such as deep learning require high computation power and also high data bandwidth which may not be possible by sticking to just either of the approaches. Thus, it has become the need of the hour to design an accelerator which achieves both these requirements.

In this project we present the SISCA architecture, for reducing the inefficiencies of moving data and exploiting high degrees of parallelism. The main contributions of the project are:

- We propose a novel SRAM based NN accelerators that enables in-place computations within SRAM sub-arrays, without transferring data in or out of it.
- The proposed architecture is build upon the idea of bit-line computing. We modify the SRAM architecture thereby allowing it to performs dot product operations.
- WE built a hierarchical memory architecture with eDRAM in the next level of memory hierarchy to address the memory limitation of SRAM.

II. Background

A. CNN/DNN

CNNs and DNNs are the most popular state of the art machine learning algorithms. These algorithms are made of large number of layers and these layers are executed sequentially. CNNs and DNNs predominantly have 3 main kind of layers: convolutional, pooling and classifier layer. Usually these layers are followed by non-linear functions such as sigmoid/ReLU/PReLU. Convolutional layers usually consist of shared kernels. In classifier/fully-connected layers, each neuron from the input feature map is connected to all the neurons in the output feature map leading to private weights for each edge.

B. SRAM Architecture & Working

SRAM (Static Random-Access Memory) is a semiconductor memory in which the data stores is static i.e., stored data can be retained indefinitely, as long as the power supply is on without any need for periodic refresh operation. The one-bit memory cell of SRAM consists of a simple latch circuit. Depending on the preserved state of the two inverter latch circuit, the data being held in the memory cell will be interpreted either as logic 0 or logic 1. To access the data contained in the memory cell via a bit line, switch is controlled by corresponding word line. The conventional 6T SRAM model can be seen in Figure 1.

1) Read/Write cycle: The read cycle is started by precharging both bit lines BL and BL_bar, i.e., driving the bit lines to a threshold. Then asserting the word line WL enables both the access transistors M5 and M6, which causes the bit line BL voltage to either slightly drop (bottom NMOS transistor M3 is ON and top PMOS transistor M4 is off) or rise (top PMOS transistor M4 is on). The BL and BL_bar lines will have a small voltage difference between them. A sense amplifier will sense which line has the higher voltage and thus determine whether there was 1 or 0 stored. The higher the sensitivity of the sense amplifier, the faster the read operation.

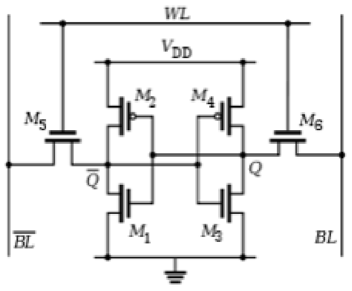

Fig. 1. 6T SRAM Cell architecuture

Write cycle begins by applying the value to be written to the bit lines. WL is then asserted and the value to be stored is latched in. As the driver of the bit lines is much stronger it can assert the inverter transistors. As soon as the information is stored in the inverters, the access transistors can be turned off and the information in the inverter is preserved.

C. Bit-line computing

SRAM memory is reconfigured to perform in-situ logical operations between two or more operands within the memory array [3]. The word line is split into two different word-lines (say wordline right (WLR) and wordline left (WLL)). This creates two independent access transistors and incurs no area overhead. When multiple word-lines are activated simultaneously in an SRAM, the shared bit-lines can be sensed to produce the result of AND (if WLRs are activated) and NOR (if WLLs are activated) on the data stored in the two activated rows.To allow such an operation, both BL and BLB are sensed separately using two single- ended SAs that are logically ANDed to indicate a match in the column. The functioning is explained in Figure 2.

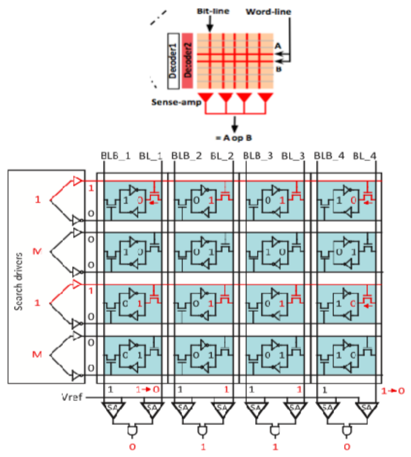

Fig. 2. Operation of bit-wise AND in SRAM

III. Related work

Lots of designs that accelerate NN applications with various platforms (ASIC, GPU, FPGA) have been proposed. DaDianNao [1] uses 36MB on-chip memory to store weights and uses NFU to perform all the computations without moving the weights around.

Recently plenty of work has carried out which use PIM and in-situ computations in accelerators. The paper of compute cache [4] expanded the idea of bit-line computing to perform computations in cache to accelerate SIMD applications. This reduces the overhead due to data movement between caches and cores. Also in-place computing avoids data transfer between caches sub-array and its controller. DRISA [5] used DRAM to accelerate applications by provide better in-situ computational performance and large memory capacity. But DRAM memory in DRISA cannot be used as conventional memory due to the modifications in the architecture. It is worth noting that all their analysis uses 1-bit weight and 8-bit activations for all its computation. ISAAC [6] used memristor crossbar array for storing the synaptic weights to perform the dot-product computation. But it might suffer from issues during fabrication due to its analog domain implementation and novel memristor crossbar arrays.

IV. SISCA Architecture

SISCA extends the idea of bit-line computing. We exploit massive data-parallelism by computing concurrently on SRAM blocks consisting of hundreds of sub-arrays distributed across different banks. The sub-arrays in an SRAM can potentially compute concurrently on data stored in them with little extensions to the existing cache structures. In addition to AND, and NOR, we extend the circuit to support XOR, and partial sum accumulations. We implement two decoders per subarray instead of one to allow activation of multiple rows. To reduce area overhead column multiplexing is implemented over the XOR reduction trees.

The basic design of SRAM is not modified. Hence, there is no extra overhead for read/write operations on SRAM using this technique. This implies that we have the flexibility to use the given SRAM as a memory unit or an accelerator depending on the requirement.

The weights and inputs are distributed across the entire SRAM array. In-situ computations are possible only when the two operands share the same bit-line. To facilitate this bit line sharing, weights and activations are always placed in two different rows of a subarray. This allows bit-line sharing between weights and activations. Each row of the SRAM array is either assigned to weights or activations. This distribution ratio is dependant on the layer of operation. If the layer is dominated by activations, then more rows are allocated to activations, and less rows to kernel and viceversa. All the inputs get multiplied to almost all the kernels to create the output feature map. Hence once all the weights in one column (16-bit partition of the cache block in one subarray) get multiplied with the activations in that column, the weights are reordered in the SRAM subarray (or across multiple subarrays or banks) to facilitate computation of the weights with the remaining activations.

Energy consumed in performing a logical computation between two rows in a sub array is assumed to twice the energy consumed in accessing one block of data from the subarray. As all the computations are performed across multiple rows of the subarray, the h-tree energy and latency does not come into picture during the computations. As H-Tree consumes nearly 90% of cache energy, this energy can be saved during computations. Although the energy of H-Tree has to be considered while moving the data between multiple subarrays.

We used 16-bit weights and activations for our analysis. As implementing in-situ multiplication operations directly over SISCA incurs huge area overhead, we resort to emulation of multiplication operation by accumulating partial sums from all the bit-wise XOR and AND combinations. So it would take us 16 iterations to create all the partial sums that would give us the result of a single MAC operation. Instead we distribute all the 16 combinations by duplicating the inputs across multiple columns thereby giving us the result of a single MAC in one cycle.

As the main purpose of this architecture is to perform massively parallel computations, we increase the row size of the sub array by decreasing the number of rows thereby keeping the size of the subarray constant. To reduce the subarray energy overhead, we have increased the number of sub arrays (this further reduces the number of rows per sub array). This helps in getting lesser latency and energy consumption. But it incurs an area overhead due to increase in the number of sub-decoders and H-Tree size.

SRAM is limited by memory capacity due to its area constraints. So we employ a hierarchical memory architecture with a combination of SRAM and eDRAM. eDRAM stores all the weights, and prefetches the weights to the SRAM arrays. Additionally, the data need not be transferred out of SRAM to apply the activation function (ReLU). It can be implemented by checking the sign bit of the data, and overwriting the data to zero if sign bit is 1.

V. Methodology

We have performed a design space exploration designing the SRAM to come up with an optimal structure for area, performance and energy efficiency. We evaluated and compared various SRAM designs for the aforementioned parameters.

A. Energy, latency and area Models

We used CACTI 6.5 [2] at 32nm technology to model the area, latency and energy models of SRAM and eDRAM. We used a ReLu activation function in our proposed design. The adders and partial sum accumulation numbers are taken directly from the analysis of DaDianNao [1]. The communications across multiple banks and subarrays is performed using a large H-Tree. The energy and area overhead of this H-Tree is obtained by modifying the CACTI source code.

The energy and area estimates for DaDianNao were directly scaled from 28nm to 32nm. This was done because their numbers of area and energy were highly efficient than the CACTI models for the same values.

B. Performance Models

We have manually mapped all our benchmark applications to DaDianNao and SISCA. We have explored various combinations to map the data so as to get the best performance comparisons between DaDianNao and SISCA. In the interest of time, we were not able to implement pipelining, pruning etc over the SISCA architecture. So the SRAM waits until all the data is loaded from eDRAM before it starts its computation. We also had an optimistic assumption that all the weights in the convolutional layer are multiplied with all the inputs in the input feature map. But in reality nearly 90% of the feature map entries only get multiplied with all the kernel entries. A similar assumption is also made

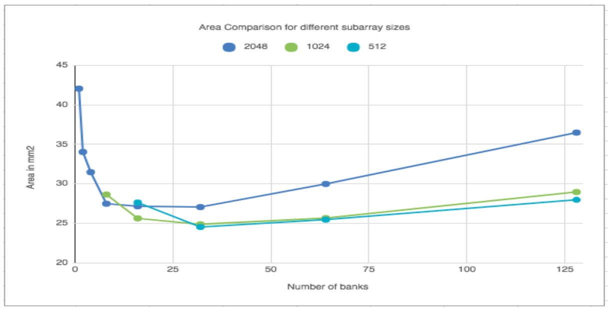

Fig. 3. Area analysis of SRAM (2 MB) with different sub array and bank sizes

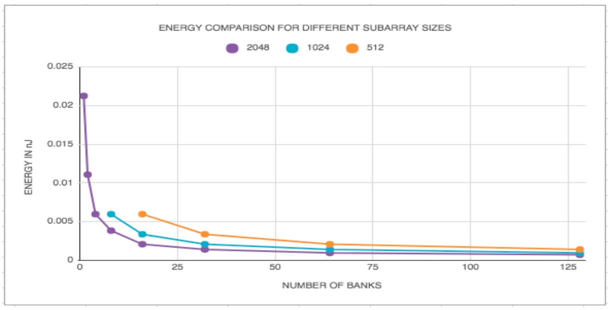

Fig. 4. Energy consumption vs Number of banks on SRAM (2 MB)

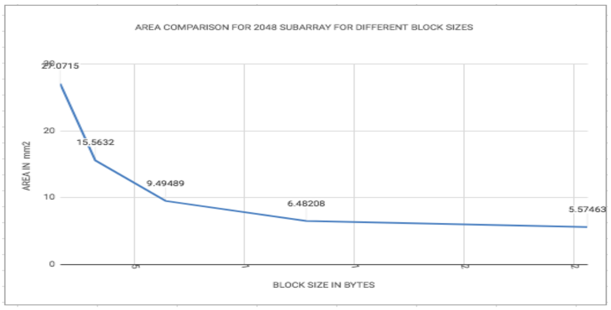

Fig. 5. Area vs Block size on an SRAM (2 MB)

in accumulating the partial products as well as off chip and on chip data movement.

C. Benchmarks

We used 3 different benchmarks to evaluate our design the ranges from a small 5 layer neural network to a deep 16 layer neural network. The largest benchmark has a total kernel memory spanning 253 MB with some layers performing more than a billion computations. One of them is LeNet5 [9], a 5 layer neural network. AlexNet [7] is an 8 layer neural network with 2 fully connected layer and a classifier layer. We also used on of the VGG [8] networks, a 16 kernel deep neural network.

VI. Experimental Results

A. Design Space Exploration

We first present the design space exploration of SRAM memory combinations we have explored. Figure 3 shows the area comparison of 2MB SRAM architecture with varying sizes of subarrays and the number of banks. Our first observation is that irrespective of the number of subarrays, the area takes a parabolic shape as the number of banks increases. Our second observation is that the minimum area is nearly the same for various subarray combinations. As our architecture focuses on massively parallelizing the computations, we have opted for a bigger subarray size (2048).

It can also be noticed from Figure 4 that energy consumption drops exponentially as the number of banks increases. The is because of the decrease in overhead on the decoders of individual banks. It has to be noted that the energy number in the graph does not include the H-Tree energy. This is because the H-Tree only comes into picture only while moving the data across and not during the computations. Although the area drops and reaches a minimum as we increase the bank count, we observe that net area is still far higher than the SRAM array with a subarray size of 512x512. This is the result of increase in the number of sub-decoders as the number of sub-arrays increases. So we experimented on varying the block size to find the ideal parameter which has an area close to the ideal area. Figure 5 shows the change in area as we vary the block size. We still kept the subarray size constant. So as we increased the block size, we reduced the number of rows per subarray thereby reducing the decoder overhead. Our design space exploration shows that a subarray of size 2048x4 is optimal in terms of energy, area and latency.

One node of SISCA consists of 14MB SRAM, and 12 MB eDRAM. Each node can perform 114688 partial products simultaneously. We have assumed 20% overhead due to the additional units over SRAM. This consumes around 65 mm^2 area in 32 nm technology.

B. Comparison to DaDianNao

The architecture is highly efficient for Convolutional layer. But it performs poorly on Fully connected layers because we are limited by how long it takes to bring these weights from an external device (eDRAM). This problem can be alleviated by introducing techniques like Deep Compression. It has to be noted that all the results shown only consider the convolutional layers, and not fully connected layers.

We have explored various combinations of block sizes to calculated the performance and energy efficiency. Figure 6 shows the energy, throughtput and area of SISCA with block sizes of 64 and 256 bytes against the DaDianNao Architecture while running the benchmark applications. It can be observed that the performance

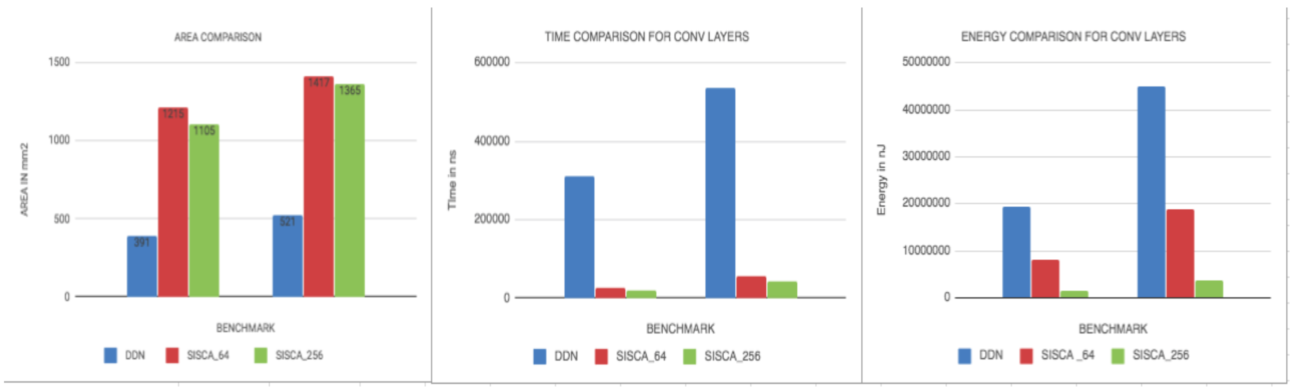

Fig. 6. Comparison of DaDianNao with SISCA at block sizes 64 and 256 bytes for Area, Throughput and energy consumption

doesn't vary significantly with the change in the block size. But there is a huge improvement in energy efficiency. This is because, as the number of sub arrays increases, the energy overhead per decoder reduces. Our results show that SISCA with a block size of 64 bytes gives throughput and energy improvement of 9.5x and 2.5x over the benchmarks against DaDianNao. Whereas SISCA with a block size of 256 bytes gives 13x and 12x improvement on throughput and energy efficiency.

VII. Conclusion

In this paper, we propose SISCA architecture that unlocks the massive bit-line computational capability of SRAM to accelerate neural networks. This allows us to do computations in-situ memory without modifying the memory's basic functionality of read & write. This helps us avoid the data movement between memory and computational units. We demonstrate the efficacy of our architecture using a suite of data intesive benchmarks. We observe that SISCA is able to outperform DaDianNao at convolutional layers. While the fully connected layers suffer from data bandwidth limitation from external eDRAM. On average, SISCA is 13x faster and 12x energy efficient than DaDianNao.

References

[1] Y. Chen et al., "DaDianNao: A Machine-Learning Supercomputer," 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, Cambridge, 2014, pp. 609-622.

[2] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0, in Proceedings of MICRO, 2007.

[3] 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," in IEEE Journal of Solid-State Circuits, vol. 51, no. 4, pp. 1009-1021, April 2016.

[4] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw and R. Das, "Compute Caches," 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), Austin, TX, 2017, pp. 481-492.

[5] Shuangchen Li, Dimin Niu, Krishna T. Malladi, Hongzhong Zheng, Bob Brennan, and Yuan Xie. 2017. DRISA: a DRAM-based Reconfigurable In-Situ Accelerator. In Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-50 '17).

[6] A. Shafiee et al., "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, 2016, pp. 14-26.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'12), F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.), Vol. 1. Curran Associates Inc., USA, 1097-1105.

[8] K. Simonyan and A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv preprint arXiv:1409.1556, 2014.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998.