SURVEY REPORT

Abstract

A generalization has been presented by researchers to address the memory wall by building a computation in-place architecture repurposing the huge cache structures to transform them into massively parallel compute units capable of running inferences for bit-line computing. Traditionally, memory wall has been addressed by building deep memory hierarchy and processing-in-memory (PIM). However, PIM and other data-intensive systems move the compute logic closer to the memory and the memory itself isn't optimized for logic computations. Thus, recent technology challenges in memories, along with an increased demand for memory capacity and performance, have fueled an active interest in alternative memory technologies. The growth in data processed and increase in the number of cores place high demands of memory systems of modern computing platforms. In-memory computing is a promising approach to addressing the processor-memory data transfer bottleneck in computing systems.

In this survey report, the above problem is addressed using various techniques: ***Neural Cache,*** a bit-serial in-cache accelerator for Deep Neural Networks; a high-performance bit-line SRAM circuit technology that enables in-place computation in caches called ***Compute Cache; DeDianNao,*** a custom multi-chip machine-learning architecture that enable high-degree parallelism at a reasonable are cost; ***Eyeriss***, A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks. To give an example of what is going to be covered, solutions to satisfy new constraints imposed by CC such as operand locality, inference throughput enhancement through Neural Cache are discussed.

1. Introduction

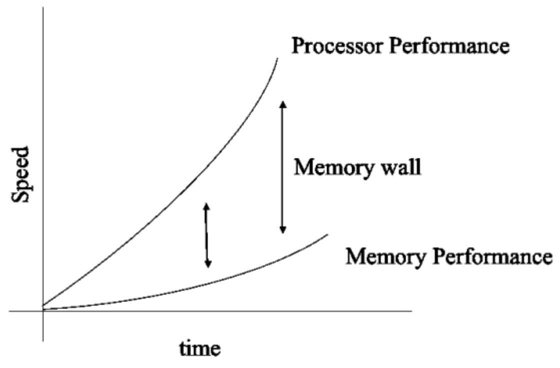
In the last few decades, the number of processor cores per chip has steadily increased while the memory latency has remained relatively constant. It can be seen from Figure 1 that instructions per second and memory capacity increases two times every two years whereas memory latency 1.1 times every two years increasing the gap between processor performance and memory performance over time leading to the **memory wall** [1]. With the advent of data intensive systems, a large amount of energy is spent in moving the data back-and-forth between memory and compute units. At the same time, neural computing and other data intensive computing have emerged exposing much higher levels of Data parallelism.

These synergistic trends have been opportunistically leveraged the huge caches present in modern processors to perform massively parallel processing for neural computing. Efficiency of these techniques arises from two main sources: massive parallelism and reduced data movement.

Bit-line computing in SRAMs has been used to implement custom accelerators: approximate dot products in analog domain for pattern recognition and CAMs. Figure 2 shows SRAM circuit for in-place computations. Two rows (WLi and WLj) are activated. An AND operation is per- formed by sensing bit-line (BL). All the bit-lines are initially pre-charged to '1'. If both the activated bits in a column have a '1' (column 'n'), then the BL stays high and it is sensed as a '1'. If any one of the bits were '0' it will lower the BL voltage below Vref and will be sensed as a '0'. A NOR operation can be performed by sensing bit-line bar (BLB).

Data corruption due to multi-row access is prevented by lowering the word-line voltage to bias against the write of the SRAM array. Measurements across 20 fabricated 28nm test chips demonstrate that data corruption does not occur even when 64 word-lines are simultaneously activated during such an in-place computation. Compute cache however only needs two. Monte Carlo simulations also show a stability of more than six sigma robustness, which is considered industry standard for robustness against process variations.

The robustness comes at the cost of increase in delay during compute operations. But they have no effect on conventional array read/write accesses.



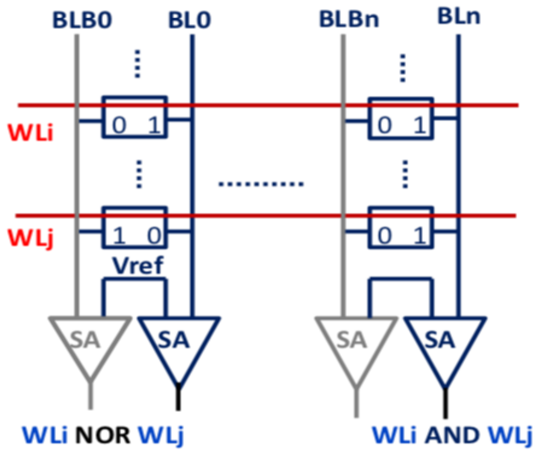


Figure 1: *Memory wall (speed vs time)*

Figure 2: *SRAM circuit for in-place computations*

This survey discusses the various techniques used to implement in-place computations. This model is particularly useful in scenarios where the same operation is applied to the same bit of multiple data elements. The key purpose of **Neural Cache** [2] is to repurpose cache structures into massively parallel compute units capable of running inferences for Deep Neural networks.

A number of neural network accelerators have been recently proposed which can offer high computational capacity/area ratio, but which remain hampered by memory accesses. he CNNs and DNNs memory footprint, while large, is not beyond the capability of the on- chip storage of a multi-chip system. This property, combined with the CNN/DNN algorithmic characteristics, can lead to high internal bandwidth and low external communications, which can in turn enable high-degree parallelism at a reasonable area cost. **DaDianNao** [3] introduces a multi-chip machine learning architecture along those lines.

Eyeriss [4] proposes RS dataflow that can adapt to different CNN shape configurations and reduces all types of data movement through maximally utilizing the processing engine (PE) local storage, direct inter-PE communication and spatial parallelism.

Compute Caches [5] use emerging bit-line SRAM circuit technology to re- purpose existing cache elements and transforms them into active very large vector computational units. Also, it significantly reduces the overheads in moving data between different levels in the cache hierarchy.

2. Background

There are some paradigms for offering bit-line computing to implement custom accelerators. Bit-serial computing in SRAM arrays can be realized by storing data elements in a transpose data layout. Transposing ensures that all bits of a data element are mapped to the same bit line, thereby obviating the necessity for communication between bit lines. The transpose data layout can be acknowledged in the following ways. In the first place, influence programmer backing to store and access information in the transpose position. This choice is helpful when the information to be worked on does not change at runtime. We use this for channel loads in neural systems. Notwithstanding, this methodology builds programming multifaceted nature by expecting programmers to reason about another data format and cache geometry.

Second, structure a couple of hardware transpose memory units (TMUs) set in the cache control box (C-BOX). A TMU takes information in the bit-parallel or standard spread out and changes over it to the transpose format before putting away into SRAM arrays or the other way around while perusing from SRAM arrays.

The second alternative is appealing on the grounds that it bolsters dynamic changes to information. TMUs can be worked out of SRAM arrays with multi-dimensional access (i.e., get to information in both horizontal and vertical directions).

Figure 3 demonstrates a conceivable TMU configuration utilizing a 8T SRAM array with sense-amps in both horizontal and vertical headings. Contrasted with a gauge 6T SRAM, the transposable SRAM requires a bigger bit-cell to empower read/write in the two bearings. Only a few TMUs are needed to saturate the available interconnect bandwidth between cache arrays. In essence, the transpose unit serves as a gateway to enable bit-serial computation in caches.

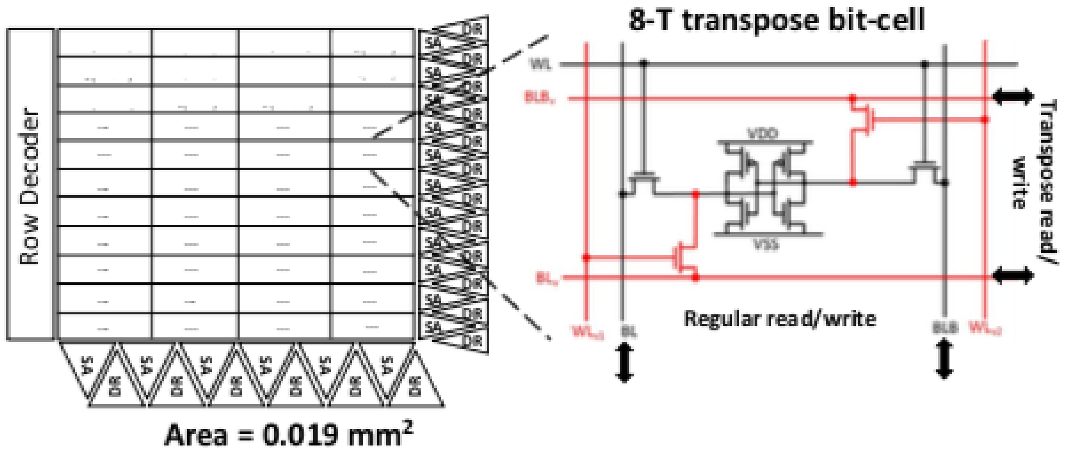


Figure 3: *Transpose Memory Unit*

The primary focus of these methodologies is in Convolutional Neural Networks that is used for visual recognition tasks. Convolution layers accounts 90% of inference operations. A convolution layer performs a convolution on the filter and the input image as can be seen in Figure 4. The dimensions of the filter are Height(R), Width(S), Channels(C), Batch of 3D filters(M) and the input image are Height(H), Width(W) and channels. Each pixel of the input is multiplied by the corresponding filter pixel across M dimensions. (RxS) Results are accumulated together and channel is reduced to a single element.

Each window of the output is of dimension 1xMxM and is slid using a stride. As the stride increases the number of output elements to be computed is reduced. The output image is based on the height, width and the stride.

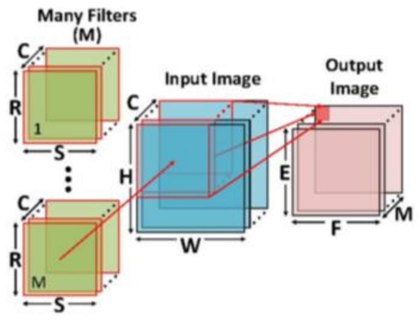


Figure 4: *Computation of a convolution layer*

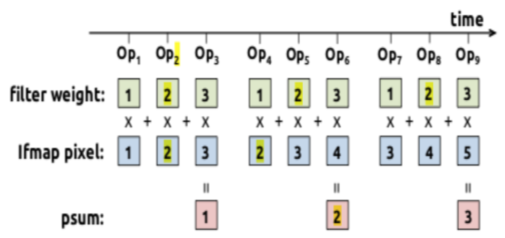

time
Op1 Op2 Op3 Op4 Op5 Op6 Op7 Op8 Op9
filter weight: 1 2 3 1 2 3 1 2 3
Ifmap pixel: 1 2 3 2 3 4 3 4 5
psum: 1 2 3

Figure 5: *Processing of a 1D convolution primitive*

The usage of the RS dataflow in **Eyeriss** is enlivened by applying a strip-mining strategy in a spatial architecture [6]. It separates the high-dimensional convolution into 1D convolution natives that can keep running in parallel; each primitive works on one column of channel loads and one line of ifmap pixels, and produces one line of psums. Psums from various natives are additionally gathered together to produce the ofmap pixels. The contributions to the 1D convolution originate from the capacity pecking order.

Each primitive is mapped to one PE for preparing; in this manner, the calculation of each line pair remains stationary in the PE, which makes convolutional reuse of channel loads and ifmap pixels at the RF level. A case of this sliding window preparing is appeared in Figure 6(a). However, since the whole convolution ordinarily contains countless natives, the accurate mapping of all natives to the PE exhibit is non-inconsequential and will incredibly influence the vitality effectiveness.

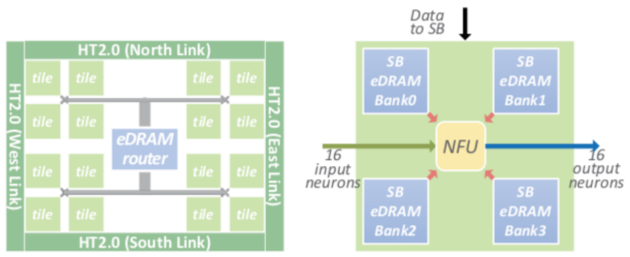


Figure 6(a): *Tile-based organization of a node and tile architecture.*

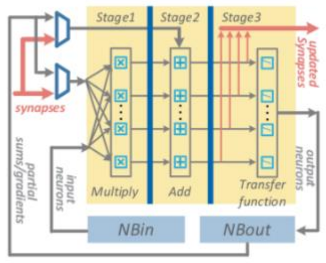


Figure 6(b): *The different operators of an NFU*

The output neurons are spread out in the different tiles, so that each NFU can simultaneously process 16 input neurons of 16 output neurons (256 parallel operations), see Figure 6(b). As a result, the NFU in each tile is significantly smaller, and only 16 × 16 × 16 = 4096 bits must be extracted each cycle from the eDRAM. We keep the 4-bank (4096-bit wide banks) organization to compensate for the aforementioned eDRAM weaknesses, and we obtain the tile design of Figure 5. We placed and routed one such tile, and obtained an area of 1.89 mm2, so that 16 such tiles account for 30.16 mm2, i.e., a 28.5% area reduction over the previous design, because the routing network now only accounts for 8.97% of the overall area.

All the tiles are connected through a fat tree which serves to broadcast the input neurons values to each tile, and to collect the output neurons values from each tile. At the center of the chip, there are two special eDRAM banks, one for input neurons, the other for output neurons. It is important to understand that, even with a large number of tiles and chips, the total number of hardware output neurons of all NFUs, can still be small compared to the actual number of neurons found in large layers. As a result, for each set of input neurons broadcasted to all tiles, multiple different output neurons are being computed on the same hardware neuron. The intermediate values of these neurons are saved back locally in the tile eDRAM. When the computation of an output neuron is finished (all input neurons have been factored in), the value is sent through the fat tree to the center of the chip to the corresponding (output neurons) central eDRAM bank.

A node contains 16 tiles, two central eDRAM banks and fat tree interconnect; a tile has an NFU, four eDRAM banks and input/output interfaces to/from the central eDRAM banks.

3. Schemes

3.1. DiDianNao

The four types of layers, pooling layers (POOL), convolutional layers, classifier layers and local response normalization layers, are run in series to form a CNN or a DNN. The DaDianNao architecture contains a Neural Functional Unit which is a pipelined adaption of calculations which are performed to assess a neuron yield. In addition to the NFU, there are buffers which are used to store input/output neurons and synapses. A neuron yield is the increase in synaptic values by neurons (input) in the regards of the primary stage and amplifications of every one of these in the second stage (adder trees), and use of a function (usually a transfer function) in the third stage (through linear interpolation). Distinctive computational operated are invoked at every stage liable to the layer type.

This architecture results in a multi-chip design for the best in class AI algorithms like CNNs and DNNs. Such algorithms derive the best speedups and area savings on both GPUs and as of late proposed accelerator. The disadvantage however is that they remain bandwidth limited.

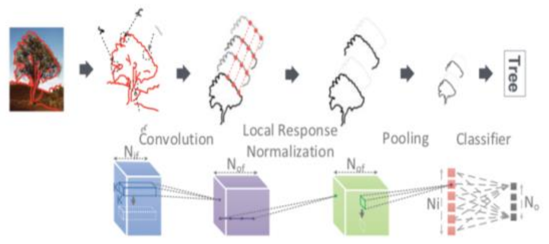


Figure 7: *The four-layer types found in CNNs and DNNs.*

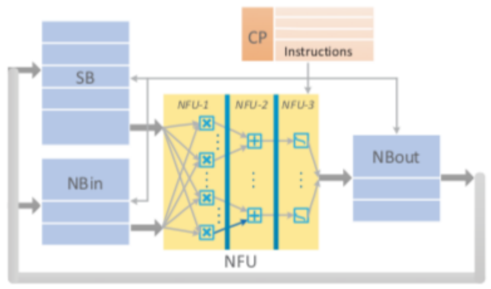


Figure 8: *Block Diagram of DiDianNao accelerator*

The design principles below are adopted: 1) The architecture should ensure that the synapses are always stored close to the neurons which will use them. This minimizes data movement which in turn saves both time and energy and there is no main memory in the architecture. 2) The architecture is asymmetric where each node footprint is biased towards storage rather than computations. 3) Neurons are transferred rather than synapses values. The reason behind this idea is that neurons are orders of magnitude fewer than the synapses in the layers mentioned above, requiring comparatively little external bandwidth. 4) High internal bandwidth is enabled by disrupting the local storage into many tiles.

3.2. Eyeriss

CNNs are constructed by stacking multiple computation layers as a directed acyclic graph. It is in the convolutional layers that the primary computations such as high dimensional convolutions take place. Matrices of ofmaps, ifmaps, filters and biases are denoted by O, I, W and B. The given stride size is denoted by U. The figure illustrates the high-level block diagram of the accelerator system which is used for the processing of CNN. The architecture of the system consists of a SA accelerator and off-chip DRAM. The inputs can be off-loaded from the CPU or GPU to DRAM and processed by the accelerator. The outputs are then written back to DRAM and further interpreted by the main processor.

The SA accelerator is composed of a global buffer and an array of PEs. Communication between the DRAM, global buffer and PE array are offered through the input and output FIFOs (iFIFO/oFIFO). This implementation of the RS dataflow is inspired by the idea of applying a strip-mining technique in a spatial architecture and breaks the high-dimensional convolution down into 1D convolution that can run in parallel.

So, each primitive operates on one row of filter weights and one row of ifmap pixels and generates one row of psums. Primitive mapping has two steps: logical mapping and physical mapping.

The logical mapping deploys the primitives into a logical PE array, the result of which has the same size as the number of 1D convolution primitives and is usually much larger than the physical PE array in hardware. The physical mapping then folds the logical PE array, so it fits into the physical PE array.

In order to maximize energy efficiency, all types of data movement are optimized by the RS by maximizing the usage of the storage hierarchy (all the way from the low-cost RF to the higher-cost array and global buffer). In the architecture, the dataflow for global multi- cast NoCs for the ifmaps and filters, and a local PE-to-PE NoC for the psum is handled using separate NoCs.

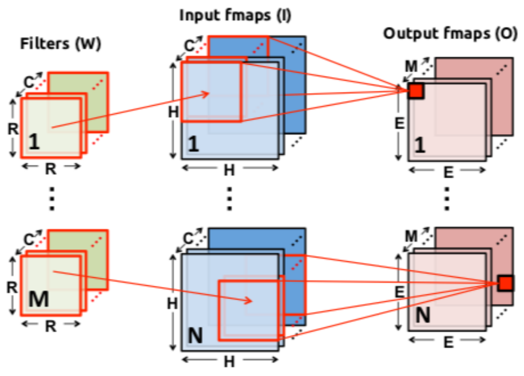


Figure 9: *Computation of a CONV/FC layer*

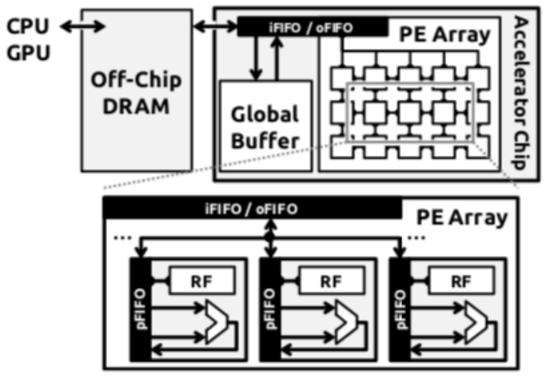


Figure 10: *Block Diagram of Spatial architecture*

3.3. Compute Cache

Figure 11 (a) illustrates a multi-core processor modeled loosely after Intel's Sandybridge. It has a three-level cache hierarchy comprising of private L1 and L2, and a shared L3. The shared L3 cache is distributed into slices which are connected to the cores via a shared ring interconnect. A cache consists of a cache controller and several banks ((Figure 11 (b)). Compute Caches use emerging bit-line computing technology in SRAMs. A Compute Cache re-purposes the elements used in this large area into compute units for a small area overhead.

A Compute Cache re-uses the elements in this large area into compute units for a small area overhead. SRAM's subarray design facilitated the in-place computation for Compute Caches. In addition to AND /NOR operations, the circuit is extended to support xor operation by NOR-ing bit-line and bit-line complement. Compound operations such as compare, and search are realized through the results of bitwise XOR.

Cache controllers are extended to provision for CC controllers which orchestrate the execution of CC instructions. The CC controller breaks a CC instruction into multiple simple vector operations whose operands span at most a single cache block and issues them to the sub-arrays. Since a typical cache hierarchy can have hundreds of sub-arrays (16MB L3 cache has 512 sub-arrays).

Provision for CC controllers are realized through the extension of Compute Cache which orchestrate the execution of CC instructions. The CC controller splits a CC instruction into multiple simple vector operations whose operands span at most a single cache block and issues them to the sub-arrays (16MB L3 cache has 512 sub-arrays).

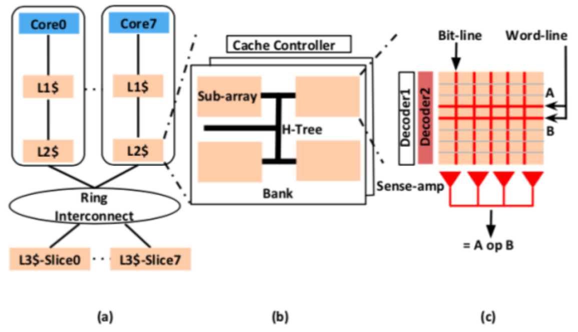


Figure 11: *Compute Cache overview. (a) Cache hierarchy. (b) Cache Geometry (c) In-place compute in a sub-array.*

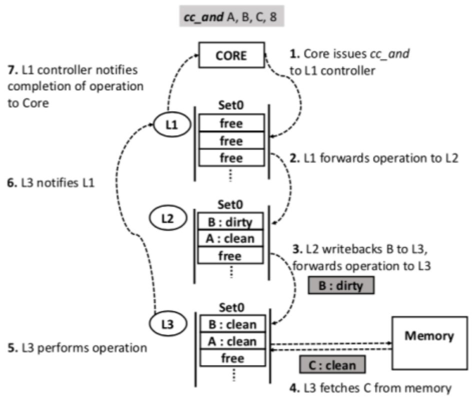

cc_and A, B, C, 8
CORE
1. Core issues cc_and to L1 controller
7. L1 controller notifies completion of operation to Core
L1
Set0
free
2. L1 forwards operation to L2
6. L3 notifies L1
L2
B : dirty
A : clean
3. L2 writebacks B to L3, forwards operation to L3
L3
B : clean
5. L3 performs operation
Memory
C : clean
4. L3 fetches C from memory

Figure 12: *Compute Cache in action*

3.4. Neural Cache

Compute cache support only several simple operations such as logical and copy. Which is why the concept of Neural Cache was introduced which gives support for more complex operations such as addition, multiplication, reduction. Facilitating interaction between these bit lines are what are the challenges in NC. For the same purpose, a bit-serial implementation with transposed data layout is proposed. A bit serial processor processes the arrays bit-slice by bit-slice. Storing data elements in a transpose data layout ensures bit-serial computing in SRAM arrays. Transposing ensures that all bits of a data element are mapped to the same bit line, thereby forestalling the necessity for communication between bit lines.

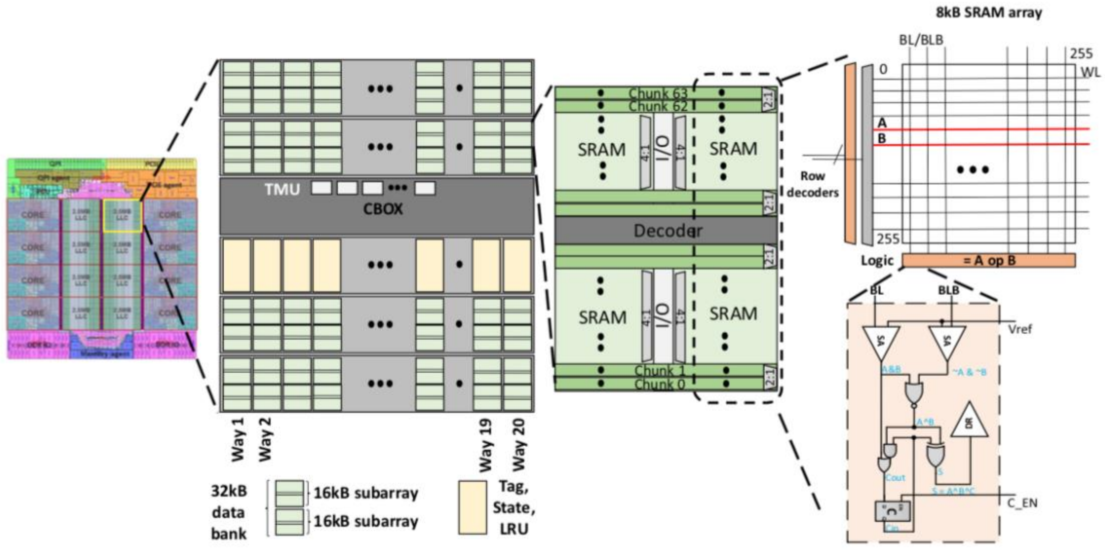


Figure 13: *Neural Cache overview.*

Figure 13 shows a Neural Cache architecture with transposed data layout. (a) Multi-core processor with 8-24 LLC slices. (b) Cache geometry of a single 2.5MB LLC cache slice with 80 32KB banks. Each 32KB bank has two 16KB sub-arrays. (c) One 16KB sub-array composed of two 8KB SRAM arrays. (d) One 8KB SRAM array re-purposed to store data and do bit line computation on operand stored in rows (A and B). (e) Peripherals of the SRAM array to support computation.

Two single- ended sense amps sense the wire-and result from two cells, A and B, in the same bitline. The sense amp in BL gives result of A & B, while the sense amp in BLB gives result of A′ & B′. Reconfigurable sense amplifier design is used which can combine into a large differential SA for speed in normal SRAM mode and separate into two single-ended SA. Area efficiency in computation mode is achieved through this purpose.

Through a NOR gate, we can get A ⊕ B which is then used to create the sum (A ⊕ B ⊕ Cin) and Carry ((A & B) + (A ⊕ B & Cin)). A 4-to-1 mux chooses the data to be written back among Sum, Carryout, Data-in, and Tag. The Tag bit is used as the enable signal for the bit line driver to decide whether to write back or not.

The Neural Cache architecture transforms SRAM arrays in LLC to compute functional units. NC requires supporting instructions such as in-cache addition, multiplication, reduction, and moves. Quantization of the outputs is done by calculating the minimum and maximum value of all the outputs in the given layer. Batch Normalization requires first quantizing to 32 bits unsigned. TensorFlow converts fully Connected layers into convolution layers. Batching is applied to increase the system throughput.

4. Experimental Results

DaDianNao achieves better clock frequency of the NFU, the multi- dimensional torus interconnects improve the scalability of large classifier layers, investigating more flexible control in the form of a simple VLIW core per node and the following results were observed. The multi-chip architecture can outperform a single GPU by up to 450.65x and reduce energy by up to 150.31x using 64 nodes. Figure 14 and 15 compare speedup and energy reduction of the multi-chip system w.r.t baseline GPU NVIDIA K20M .

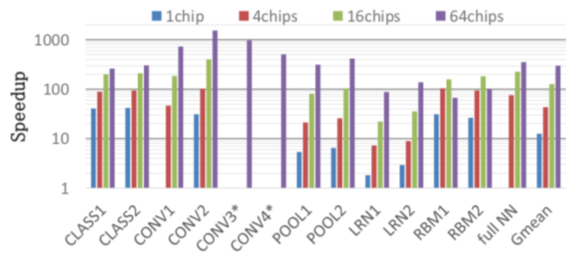


Figure 14: *Speed w.r.t. the GPU baseline*

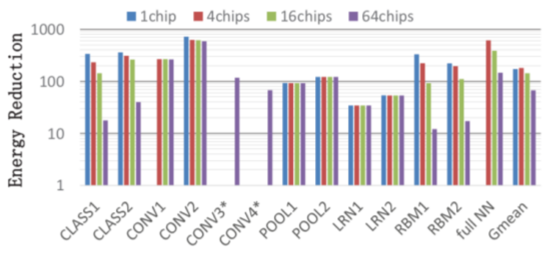


Figure 15: *Energy reduction w.r.t. the GPU baseline*

Eyeriss minimizes energy consumption by maximizing input data reuse (filters and feature maps) and minimizing partial sum accumulation cost simultaneously, and by accounting for the energy cost of different storage levels using the novel dataflow, called row stationary (RS). Experiments using the CNN configurations of AlexNet show that the proposed RS dataflow is more energy efficient than existing dataflows in both convolutional (1.4× to 2.5×) and fully-connected layers (at least 1.3× for batch size larger than 16) as can be seen from Figure 16.

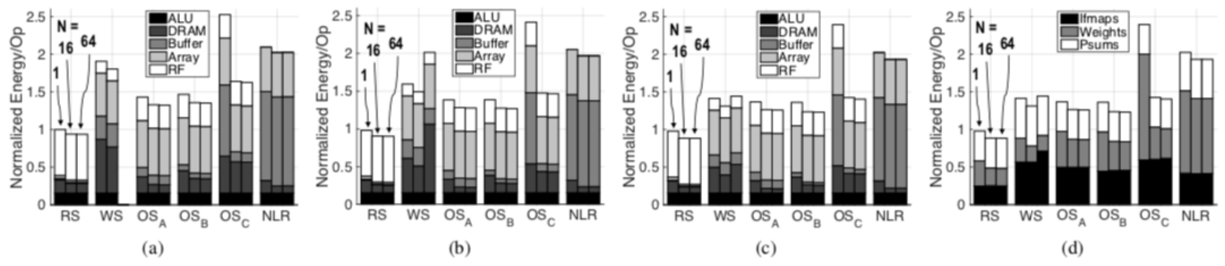
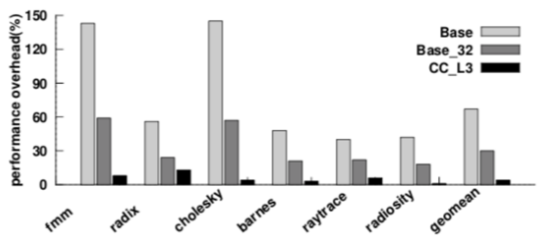


Figure 16: *Energy consumption of the six dataflows in CONV layers of AlexNet under PE array size of (a) 256, (b) 512 (c) 1024. (d) is the same as (c) but with energy breakdown in data types.*

Compute cache architecture unlocks hitherto untapped computational capability present in on-chip caches by exploiting emerging SRAM circuit technology. Using bit-line computing enabled caches, several simple operations in-place in cache over very-wide operands can be performed. This exposes massive data parallelism saving instruction processing, cache interconnect and intra-cache energy expenditure. The efficacy of Compute Caches (CC) using both micro-benchmark study and a suite of data-intensive applications can be seen in the following Figure 17 and Figure 18.



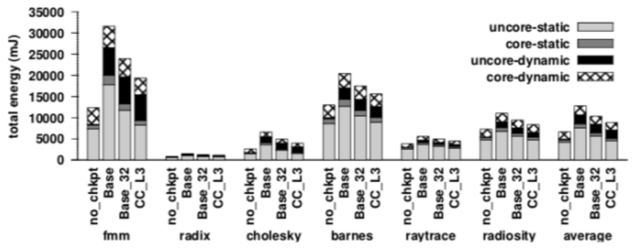


Figure 17: *Performance overhead of CC for checkpointing checkpointing*

Figure 18: *Total energy with and without*

Compute Caches increase performance by 1.9× and reduce energy by 2.4× for a suite of data-centric applications, including text and database query processing, cryptographic kernels, and in-memory checkpointing.

Neural Cache re-purposes cache structures to transform them into massively parallel compute units capable of running inferences for Deep Neural Networks. Experimental results show that the proposed architecture can improve inference latency by 18.3× over state-of-art multi-core CPU (Xeon E5), 7.7× over server class GPU (Titan Xp), for Inception v3 model. Neural Cache improves inference throughput by 12.4× over CPU (2.2× over GPU), while reducing power consumption by 50% over CPU (53% over GPU) as seen in Figure 19-23.

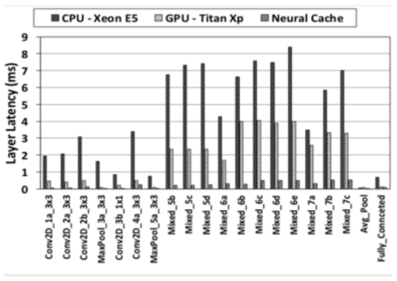


Figure 19: *Inference Latency by layer*

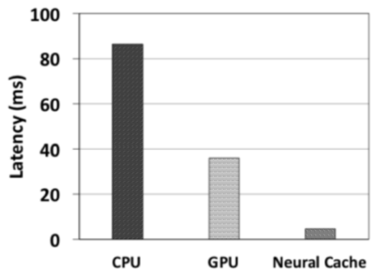


Figure 20: *Total latency*

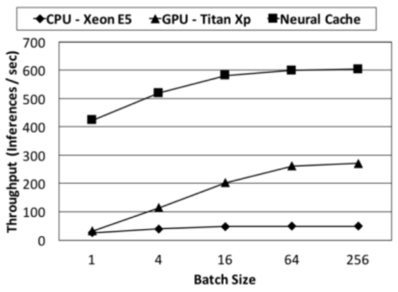


Figure 21: *Throughput with varying Batch*

Increasing the slices in Neural Cache speeds up most aspects of the inference. The total number of arrays that compute increases thus, increasing convolutions that can be done in parallel.

	CPU	**GPU**	**Neural Cache**
Total Energy / J	9.137	4.087	0.246
Average Power / W	105.56	112.87	52.92

Figure 22: *Energy Consumption and Average Power*

Cache Capacity	35MB	45MB	60MB
Inference Latency	4.72 ms	4.12 ms	3.79 ms

Figure 23: *Scaling with Cache Capacity (Batch Size = 1)*

5. Conclusion

The survey examines the weaknesses of various implementations of neural computing and other data intensive computing applications to expose much higher levels of data parallelism. The growth in data processed and increase in the number of cores place high demands of memory systems of modern computing platforms. In-memory computing is a promising approach to addressing the processor-memory data transfer bottleneck in computing systems. The operand locality constraint in these architectures are satisfied by bit-line computing where the data operands are stored in rows that share the same set of bit-lines that is to perform the computation directly on bit-lines of the memory itself, keeping data in-place. Using bit-line computing enabled caches, several simple operations in-place in cache over very-wide operands can be performed. Neural Cache imposes a dual responsibility on caches: store and compute data and turn cache architectures into massively parallel vector units, and drastically reduce on-chip data movement overhead. Another method, that relies on bit-line computing model is the Compute C ache architecture. Compute cache arithmetic and other neural network data layout solutions provides competitive performance compared to modern GPUs with negligible area overheads. DiDianNao is a multi-chip architecture that outperforms a single GPU by increasing the clock frequency of the NFU, multi-dimensional torus interconnects to improve the scalability of large classifier layers, investigating more flexible control in the form of a simple VLIW core per node and the associated toolchain. Eyeriss is an analysis framework to evaluate the energy cost of different CNN dataflows on a spatial architecture. Using the RS dataflow, the area allocation between processing and storage has a limited effect on energy-efficiency, since more PEs allow for better data reuse, which balances out the effect of less on-chip storage.

References

[1] W. A. Wulf and S. A. McKee, “*Hitting the memory wall: Implications of the obvious*,” SIGARCH Computer Archit. News, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[2] Charles Eckert, Xiaowei Wang, Jingcheng Wang, Arun Subramaniyan, Ravi Iyer, Dennis Sylvester, David Blaauw, and Reetuparna Das, "*Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks*," in Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

[3] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, and O. Temam, "Dadiannao: A machine-learning supercomputer," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2014, pp. 609–622.

[4] *Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," in Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on. IEEE, 2016, pp. 367–379.*

[5] Shaizeen Aga, Supreet Jeloka, Arun Subramaniyan, Satish Narayanasamy, David Blaauw, and Reetuparna *Das, "Compute Caches," in Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on. IEEE, 2016, pp. 367–379.*

[6] J. J. Tithi, N. C. Crago, and J. S. Emer, "*Exploiting spatial architectures for edit distance algorithms*," IEEE ISPASS, 2014.

[7] A. Parashar, M. Rhu, A. Mukkara, A. Puglielli, R. Venkatesan, B. Khailany, J. Emer, S. W. Keckler, and W. J. Dally, "*Scnn: An accelerator for compressed-sparse convolutional neural networks*," in Proceedings of the 44th Annual International Symposium on Computer Architecture. ACM, 2017, pp. 27–40.