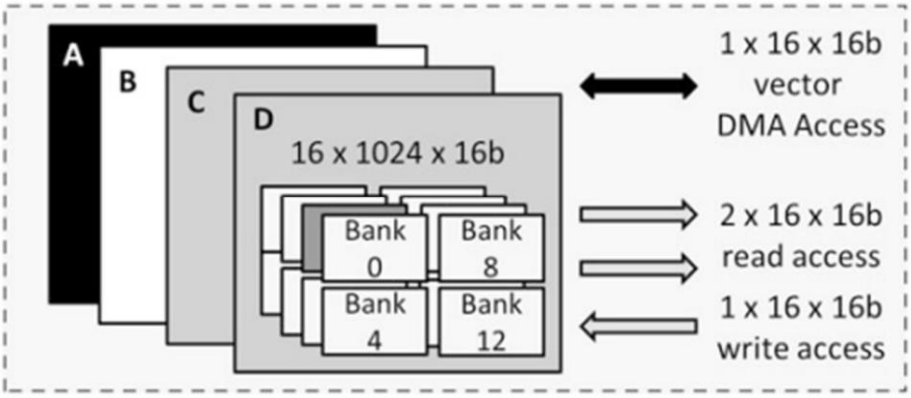
Assignment No. 8

Q1. Explain On-chip and Off-chip memory architecture. Which is faster and why? On-chip



On-chip memory architecture. One large address space is subdivided into 4 blocks, each containing 16 banks of single port SRAM. Every SRAM macro can store 1024 × 16b words. The architecture allows scalar, vector, or double vector access from the processor side, while being read or written through a vector port from the DMA side

It is organized as one large 16b memory address space (128 kB) and is subdivided into four blocks (32 kB) containing 16 single port SRAM banks (2 kB) of 1024×16b words. Every block hence allows vector access, reading/writing one word from every bank. Single word access is also possible for scalar operations and for the FIFO-based architecture. The programmer is free to store feature maps or filter bank weights in any of the four available memory blocks.

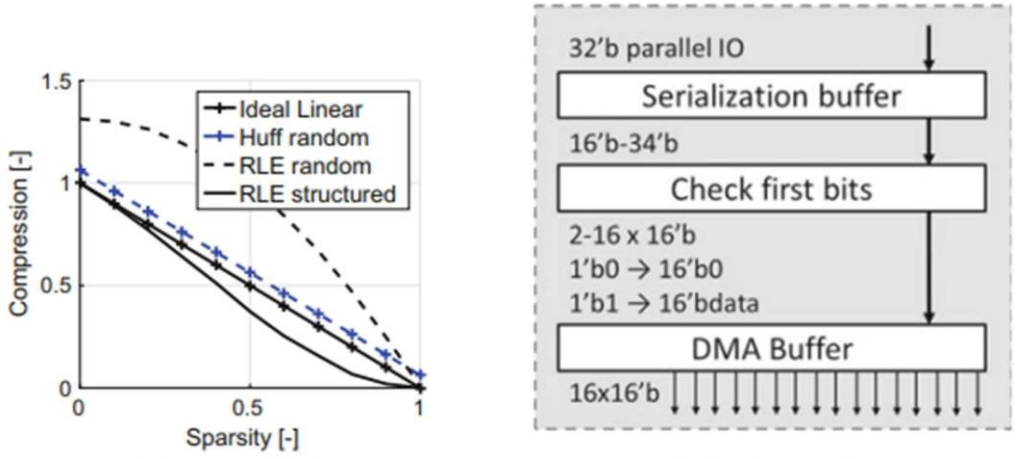
From the processor side, two of the four blocks can be read simultaneously. Typically one block would contain only filter values and a second block a part of an input feature map. This allows fetching both filter and feature inputs for the 2D MAC array in a single cycle. There is only one write port from the processor side, due to the lower amount of write accesses.

On-chip memory refers to the memory resources that are integrated directly onto the processor chip itself. This typically includes cache memory, such as L1, L2, and sometimes L3 caches. On-chip memory is physically located on the same silicon die as the processor cores.

Advantages:

- Low Latency: On-chip memory has very low access latency since it is physically close to the processor cores.
- High Bandwidth: The communication between the processor cores and on-chip memory can happen at high speeds, offering high bandwidth.
- Reduced Power Consumption: Accessing on-chip memory typically consumes less power compared to accessing off-chip memory due to shorter distances and simpler communication paths.

Off-chip



(a) RLE vs Huffman (b) Implementation

Off-chip communication is controlled through a DMA containing an en/decoder. I implement a linear compression scheme based on 2-symbol Huffman encoding. If data is zero, the 16b data word is encoded as a 1 b0. If the word is non-zero, it is encoded as a 17b word {1 b1, 16 bdata}. As shown in Fig. 5.7, this allows near ideal linear compression with the compressed data-size (C): C = (s × 1 n +(1−s)× n+1 n). Here, s is the degree of sparsity and n = 16 is the word length.

Although other available compression algorithms such as run-length encoding (RLE), used in Chen et al. (2016) e.g., could potentially achieve super-linear compression, it does not necessarily perform well on CNN data sets. This is illustrated in Fig. 5.7a, where the plotted compression rate for RLE is a function of the data distribution and clustering, while for Huffman, it is only a function of sparsity

Off-chip memory refers to memory resources that are located outside the processor chip. This includes DRAM (Dynamic Random Access Memory), SRAM (Static Random Access Memory), and other forms of external memory.

Advantages:

- Higher Capacity: Off-chip memory systems can offer significantly larger storage capacities compared to on-chip memory.
- Cost-Effectiveness: Off-chip memory is generally less expensive per unit of storage compared to on-chip memory.
- Flexibility: It's easier to upgrade or replace off-chip memory modules without having to modify the processor chip.

Q2. Explain 2D MAC processor architecture for Embedded CNN.

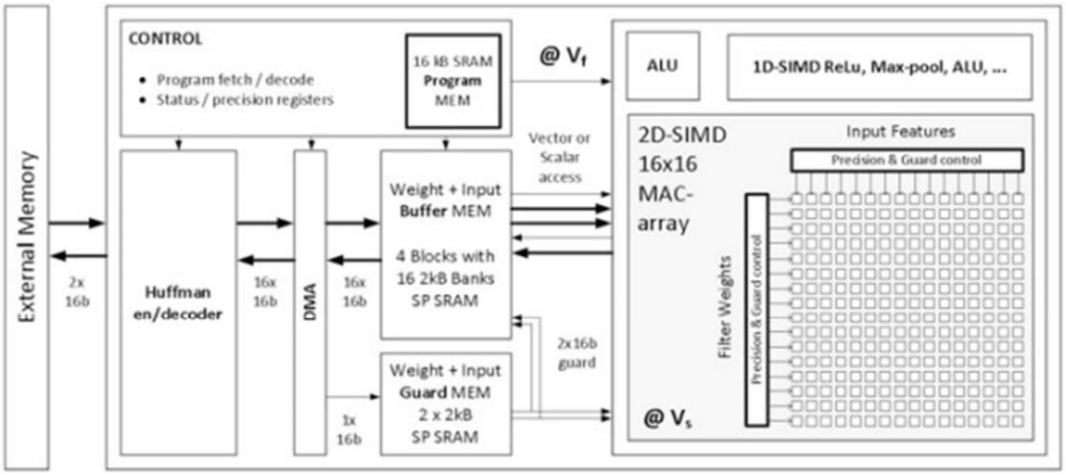


Fig. 5.2 High-level overview of the Envision architecture. The chip contains a scalar and 16× vector ALU, an input and weight on-chip SP SRAM, an SRAM storing sparsity information, a control unit, a DMA, and a Huffman-based en/decoder in a fixed power domain at supply V_f. A 2D SIMD MAC array is put in a scalable power domain at supply V_s

The 16 × 16 2D-MAC array, shown in Fig. 5.2, with its dataflow illustrated in Fig. 5.3, operates as a convolution engine. This array of single-cycle MACs achieves a 256× speedup compared to a scalar solution, while minimizing bandwidth to on chip memory. The 2D-architecture allows applying 16 different filters to 16 different units of the input feature map simultaneously, which is an effective exploitation of the convolutional and image reuse available in CNNs (Chen et al. 2016). This exploitation of data reuse thus allows a 256× speedup, at more than 16× lower internal bandwidth compared to a naive 1D-SIMD solution, requiring 2 inputs per processing unit per cycle. The local communication overhead can be further reduced by adding a FIFO register at the input of the MAC array, as shown in Fig. 5.3. Table 5.1 shows a comparison of the bandwidth fetch-reductions of the 2D-MAC array architecture compared to the naive 1D-SIMD baseline.

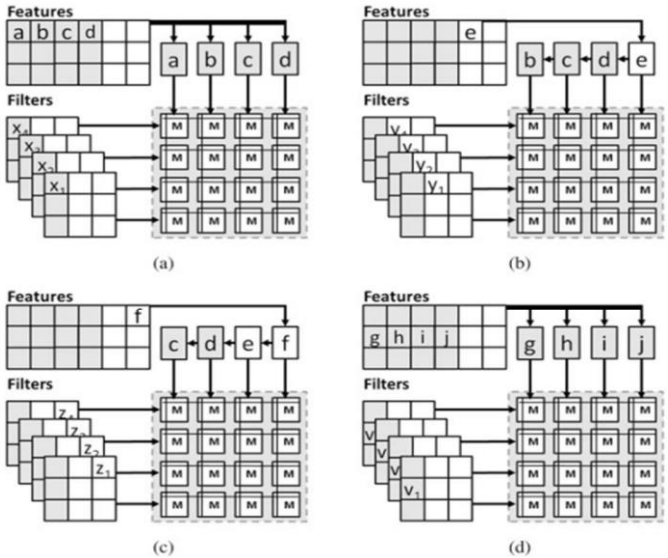


Fig. 5.3 Example of a typical CNN dataflow in Envision. **(a)** Step 1. **(b)** Step 2. **(c)** Step 3. **(d)** Step 4

The four first operation steps of a typical 3 × 3 × C × F filter dataflow, of which 4 out of F filters are performed in parallel. The concept is illustrated for a simplified 4 × 4 MAC array for clarity, but the chip has a 16 × 16 array. In this simplified example a vector is loaded from the input feature map buffer to a FIFO register as a first step. Then, it is multiplied with the first (w00) filter values of four different filters out of F. All 16 partial sums are then accumulated with the previous result and stored in a matrix of local accumulation registers. In the second and third steps, a single word is fetched from the input feature map memory and pushed through the FIFO. This shifted vector is again multiplied with the next four filter values and accumulated with the previous result. This sequence is repeated three times for the 3 × 3 filter, illustrated by step 4, in which a vector of the next input row is multiplied and accumulated with the correct weights.

An additional advantage of this scheme is that it allows all intermediate values to be kept in the accumulation registers for a full K × K × C convolution. Therefore the MAC accumulation registers are 48b, which is an over-design for the worst case in the AlexNet (Krizhevsky et al. 2012) benchmark. Because of this, there is no need for frequent write-backs to more expensive SRAM.

The presented 2D-MAC-unit with FIFO-input also supports CNN dataflows with strides not equal to 1. This is illustrated in Fig. 5.4. Figure 5.4a is a representation of the dataflow when the stride is 1 and the kernel size k×k is 3×3, similar to the flow depicted in Fig. 5.3. Figure 5.4b is a representation of a CNN-layer with k = 3 and a stride of 2. Here, both the input features and weights require re-ordering compared to the standard scheme with a stride of 1. In this scheme, a vector of 16 features is fetched first, together with w00. This vector stores features 0, 2,..., 30 of the first line. The second cycle is a single word FIFO-access, fetching feature 32 and pushing it to the FIFO, while fetching w02. Finally, this scheme requires another vector access of features 1, 3,..., 31, together with w01.

Q3. How On-chip memory architecture applied in CNN.

On-chip memory architecture plays a critical role in accelerating Convolutional Neural Networks (CNNs) by providing faster access to frequently used data and reducing reliance on slower off-chip memory. Here's how it's applied:

Bottleneck Reduction:

- Traditional CNN processing relies heavily on off-chip memory (DRAM) to store both input data (images) and filter weights.
- Accessing data from DRAM creates a bottleneck due to its slower access times compared to on-chip memory.

On-chip Memory Benefits:

- CNNs involve repeated multiplication and accumulation operations between filter weights and small image patches.
- By storing these frequently accessed elements (weights, image patches) in on-chip memory (caches or SRAM), the processor can retrieve them much faster, significantly improving overall performance.

Cache Hierarchy Exploitation:

- Modern processors employ a cache hierarchy (L1, L2, L3 caches).

- The goal is to keep the most frequently used weights and input data sections closer to the processing cores in faster caches (L1, L2).
- This reduces the need to access slower off-chip memory (DRAM) for every operation.

Data Reuse Strategies:

- On-chip memory architecture can be optimized for CNNs by implementing data reuse techniques.
- For example, exploiting spatial locality in filter weights and input features allows the processor to reuse loaded data for multiple computations within the cache, minimizing redundant fetches from off-chip memory.

Specialized Techniques:

- Some architectures incorporate specialized on-chip memory structures like weight prefetching buffers or scratchpad memory.
- These buffers can preload frequently used weights or intermediate results, further reducing reliance on off-chip memory access.

Impact on Performance:

- By effectively utilizing on-chip memory architecture, CNN processing can achieve significant speedups.
- Faster access to weights and input data translates to faster execution of convolution operations, leading to improved overall inference or training times.

Challenges and Trade-offs:

- On-chip memory has limited capacity compared to off-chip memory.
- Careful selection of data to store on-chip and efficient data management techniques are crucial for optimal performance.
- Additionally, balancing on-chip memory usage with processing core requirements is essential to avoid resource conflicts.

In conclusion, on-chip memory architecture acts as a performance booster for CNNs by providing faster access to critical data elements. By strategically utilizing caches, data reuse strategies, and specialized memory structures, CNN processing can be significantly accelerated on devices with limited resources.