Chapter 2

Analyzing Off-chip Memory Accesses
~~Analytical Framework~~

NN accelerators use on-chip memory and reuse the data from it to minimize off-chip memory accesses. Figure 2.1 illustrates the impact of on-chip memory data reuse on data access energy. Figure 2.1a shows the memory accesses and energy estimates when N bytes are directly accessed from the off-chip memory and Figure 2.1b shows the reduction in off-chip memory accesses when re-using the data from on-chip memory. If the energy per bytes access from the off-chip and on-chip memories are e_1 and e_2, respectively, the energy efficiency can be expressed as,

Needs to be elaborated. First define all the symbols and state assumptions. Then derive the expression.

$$E_{efficiency} = 1 - \frac{E_2}{E_1} = 1 - (\frac{1}{n} + \frac{e_2}{e_1}) \tag{2.1}$$

For a given architecture, e_1 and e_2 are fixed and e_1 is significantly high compared to e_2. Equation 2.1 shows energy efficiency mainly depends on data reuse and improves significantly with data reuse. DNN accelerators use multiple levels of on-chip memories and apply techniques to maximize the data reuse from the lower memories to improve energy efficiency.

The NN accelerator reads the input data (or input activations), filter weights, and partial sums from the off-chip memory and stores them temporarily in the on-chip memory. The PE-array reads the data from the on-chip memory to perform the computations and then stores them back to the on-chip memory. The partial sums or the outputs of the computations are then finally stored in the off-chip memory. The flow of the data is shown in Figure 2.2. Due to large difference between the energy consumption of accessing the data from the off-chip memory compared to access energy from the on-chip memory and computation energy, NN accelerators aims to minimize the off-chip memory accesses by exploiting the memory hierarchy and data-reuse from them.

The layer data is stored as multi-dimensional arrays in the off-chip memory, which is generally too large to fit in the local on-chip memory. In order to perform the computations,

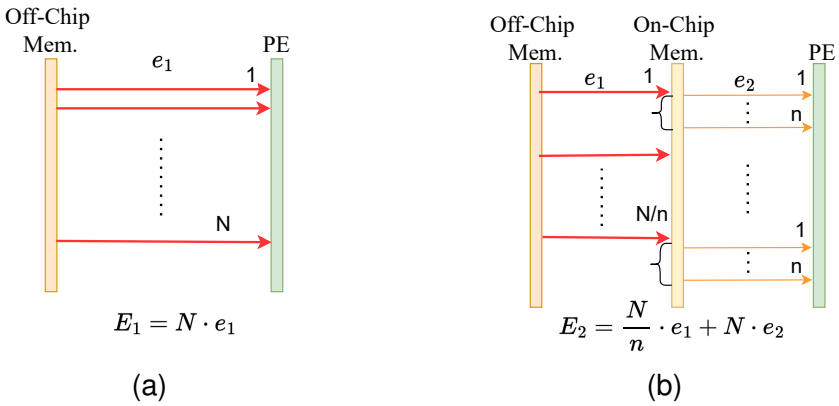


Figure 2.1: Memory accesses and energy estimates for accessing data (a) always from the off-chip memory. (b) from two-level of memory hierarchy while reusing the data from on-chip memory.

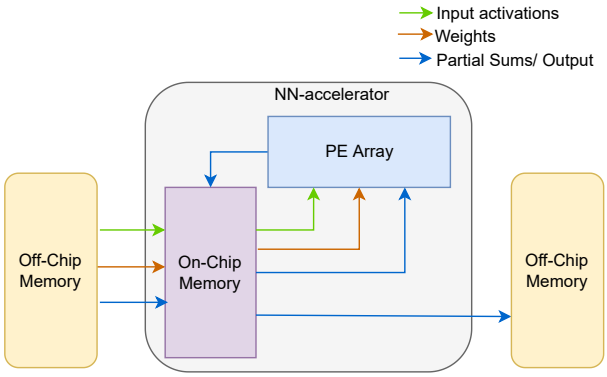
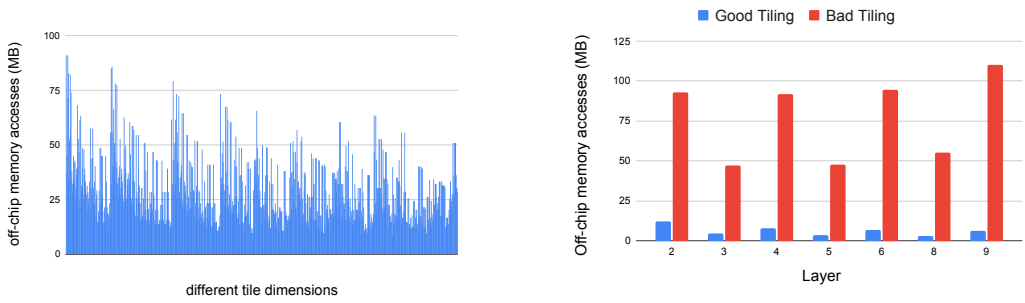


Figure 2.2: Read/Write of inputs, weights and partial/final outputs from different level of memories.

How did you get the graphs of figure 2.3?

the large layer data is partitioned into small tiles. These tiles are fetched from the off-chip memory, repeatedly to compute the final output sum. Figure 2.3 shows the impact of tile dimensions on off-chip memory accesses for a popular CNN, VGG16, for a given tile scheduling scheme. Figure 2.3a shows using different tile dimensions results in different volume of off-chip memory accesses. A good tile dimension can reduce the off-chip memory accesses upto 90% compared to a bad tile dimension as shown in Figure 2.3b for different layers of VGG16. The tile dimensions and the order in which these tiles are processed (scheduling of the tile operations) significantly impact the volume of data reuse and, thus, the overall energy consumption and throughput of DNN accelerators.

Determining the optimal tile dimensions and scheduling scheme requires comparing the off-chip memory accesses for different tile dimensions and scheduling schemes. Modern DNNs have a variety of layers (e.g., convolution, fully connected, recurrent, pooling, etc.),



(a) (b)

Figure 2.3: Off-chip memory accesses for 64KB on-chip memory (a) off-chip memory accesses using different tile dimensions for a single layer. (b) Impact of good and bad tile dimensions on different Conv. layers of VGG16.

each exhibiting different types of data access patterns. Even the layers of the same type differ in shape and size. Due to varying layer shapes, sizes and type, optimal partitioning and scheduling vary among layers. Finding the optimal partitioning and scheduling scheme, by performing the measurements on the hardware, is time-consuming, and large search space makes it practically impossible.

To address this, we have developed an analytical framework that integrates models of NN layers to compute a layer's off-chip memory accesses, data access energy and the number of compute cycles for mapping a layer on a given PE array. Figure 2.4 shows the block diagram of the analytical framework. The framework is used as a design space exploration engine to find the optimal partitioning and scheduling scheme for a given layer-type and layer-shape to optimize NN accelerators' energy efficiency and throughput.

Needs to be elaborated. First define all the symbols and state assumptions. Then derive the expression.

The off-chip memory access of a partitioned 3D data can be computed as follows

Is this equation used later?

$$\mathbb{B}_{3D} = \sum_{t=1}^{N_{tiles}} (\mathbb{B}_t \times r) \tag{2.2}$$

where N_{tiles} is the number of tiles. r and $\mathbb{B}_t$ are the trips count and the number of bytes accessed from off-chip memory for the t^{th} tile, respectively.

The analytical framework computes the off-chip memory accesses of a given 3D data using the above methodology while considering the data resolution and architectural constraint. The framework considers the bus width and data alignment to precisely compute the off-chip memory accesses. It takes the address and shape of the data as input and iterates for all the tile dimensions. Analytical framework implements models for different layer-types and data reuse schemes to precisely compute the memory accesses and data access energy.

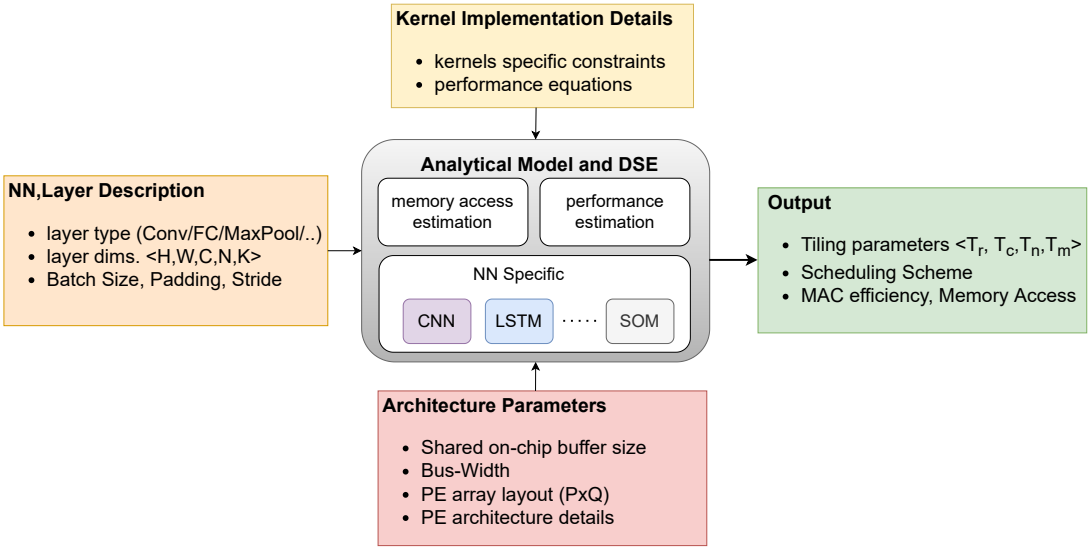
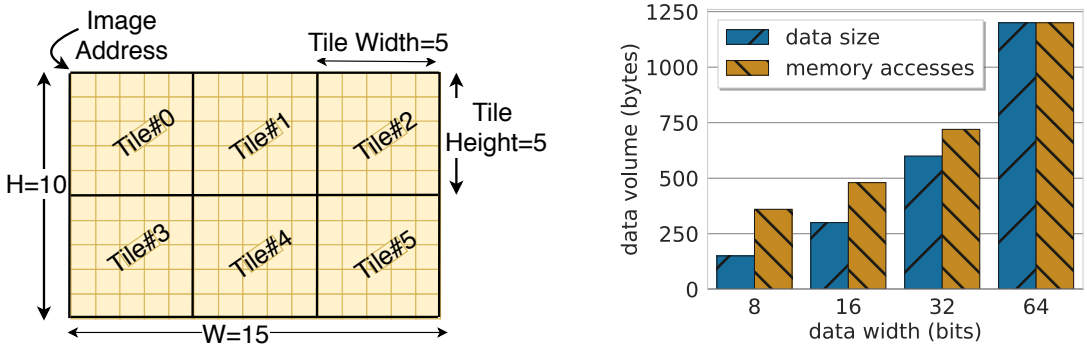


Figure 2.4: Analytical framework to estimate the performance, off-chip memory accesses and energy of DNNs.

2.1 Analytical study of architectural parameters



(a) A 2D image and its tiles

(b) Data size vs. memory accesses

Figure 2.5: A 2D data and memory accesses on 64-bit data bus

Why have you changed from 3D to 2D?

How is table 2.1 obtained?

Fig. 2.5a shows a 2D image of shape 10×15, partitioned into tiles of dimensions 5×5. Although all tiles have the same size, the off-chip memory accesses for different tiles are not the same. Table 2.1 shows the sizes and off-chip memory accesses of each tile on 64 bits wide bus for 8 bits data width. The difference between the tile sizes and the memory accesses is due to the bus width and address alignments of different rows of the tile.

Fig. 2.5b shows the size ($H \times W$) and off-chip memory accesses of the image shown in Fig. 2.5a, for different data bit widths. For smaller data bit width, the ratio of memory accesses to the data size is higher. Modern NN accelerators use wide memory bus to improve the

Table 2.1: Off-Chip memory accesses of the tiles of size 5×5

Tile#	0	1	2	3	4	5	**Total**
Size(bytes)	25	25	25	25	25	25	150
Mem. Access(bytes)	72	56	56	48	72	56	360

memory bandwith and low number of bits to represent the data to reduce the storage requirements and memory accesses. Therefore it is crucial to consider the architectural parameters for low resolution data.

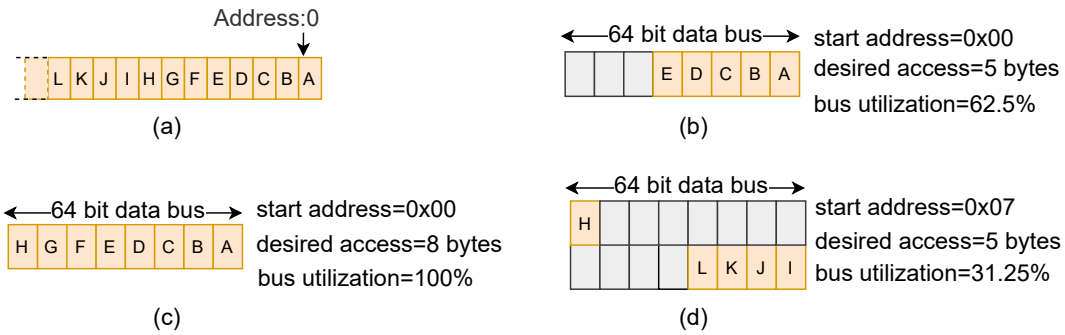


Figure 2.6: Off-chip memory accesses on 64-bit wide data bus

The DNN accelerators use a wide data bus to access off-chip memory to meet the high memory bandwidth requirement [6, 8]. If the number of bytes accessed from an off-chip memory address is not a multiple of bus width or the address is not aligned to the word boundary, it results in unused bytes lanes of the data bus. Figure 2.6 illustrates memory accesses on a 64-bit data bus. Fig. 2.6a shows a read transaction of 8 bytes from an aligned address and uses the full bus width. However, if only 5 bytes are read from an aligned address, as shown in Figure 2.6b, 8 bytes are still accessed. If 5 bytes are read from an unaligned address, it results in 16 bytes of data access, as shown in Fig. 2.6c. The unused byte lanes do not carry any useful data, but they contribute to overall energy consumption. The length of the data read should be chosen such that bus utilization is high and off-chip memory accesses and energy consumption are minimized.

This should go to section 2.1.2

To analyze the effect of bus width and address alignment, we express number of bytes accessed from off-chip memory ($\mathbb{B}$) for l bytes of data as a function of l, its off-chip memory address ($Addr$), and the bus width (BW).

2.1.1 ~~Bus Width~~ Aligned access

Number of bytes accessed from off-chip memory ($\mathbb{B}_{align}$) from aligned addresses for different data length (l) is in multiples of bus width (BW), e.g., reading 10 bytes results in 16 bytes of off-chip memory access. $\mathbb{B}_{align}$ for l bytes and bus width BW can be expressed as

$$\mathbb{B}_{align}(l, BW) = \lceil \frac{l}{BW} \rceil \times BW$$ Equation number?

2.1.2 ~~Address Alignment~~ General case

The off-chip memory bus protocol supports burst based transactions where multiple data transfers of *transfer size* happen from a starting address [3]. Most transfers in a transaction are aligned to the *transfer size*. However first transfer may be unaligned to the word boundary.

The number of bytes accessed from off-chip memory for accessing l bytes from address $Addr$ on BW bytes of bus width can be expressed as

$$\mathbb{B}(Addr, l, BW) = (\lceil \frac{Addr + l}{BW} \rceil - \lfloor \frac{Addr}{BW} \rfloor) \cdot BW \tag{2.3}$$

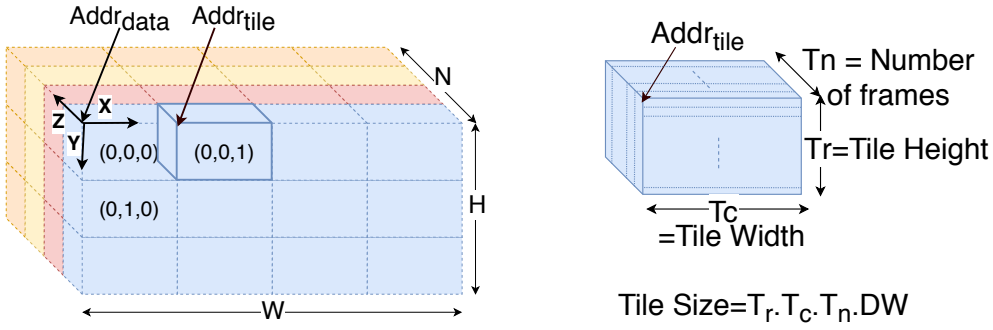
Mention that the equation of 2.1.1 is a special case of this.

2.2 Off-Chip Memory Access of 3D Data

Consider a 3D data of shape $\langle W, H, N \rangle$, partitioned into tiles of dimension $\langle T_c, T_r, T_n \rangle$ as shown in Fig. 2.7a and Fig. 2.7b. Each tile is identified by index (x, y, z) where $x, y, z \in \mathbb{Z}$ and

$$0 \leq x < \lceil \frac{W}{T_c - \delta} \rceil, 0 \leq y < \lceil \frac{H}{T_r - \delta} \rceil, 0 \leq z < \lceil \frac{N}{T_n} \rceil \tag{2.4}$$

δ is the number of elements or rows overlapping between adjacent tiles.



(a) A 3D data partitioned into tiles

(b) A 3D tile

Figure 2.7: Off-Chip Accesses and a 3D partitioned data and tiles

Algorithm 1 BW Aware off-chip memory access

1: **procedure** BWA($Addr_{data}, \langle T_c, T_r, T_n\rangle, \langle W, H, N\rangle, \delta$)
2: $Acc_{data} \leftarrow 0$
3: $Addr_{tile} \leftarrow Addr_{data}$
4: **for** $x, y, z \leftarrow (0,0,0)$ **to** $(\lceil \frac{W}{T_c-\delta} \rceil - 1, \lceil \frac{H}{T_r-\delta} \rceil - 1, \lceil \frac{N}{T_n} \rceil - 1)$ **do**
5: $c \leftarrow x \cdot (T_c - \delta)$
6: $r \leftarrow y \cdot (T_r - \delta)$
7: $n \leftarrow z \cdot T_n$
8: $Addr_{tile} \leftarrow$ GETTILEADDR$(Addr_{data}, c, r, n, \langle W, H, N\rangle)$
9: $\langle T_c^1, T_r^1, T_n^1\rangle \leftarrow$ GETTILEDIM$(c, r, n, \langle T_c, T_r, T_n\rangle, \langle W, H, N\rangle)$
10: $Acc_{data} \leftarrow Acc_{data} +$ GETTILEACC$(Addr_{tile}, \langle T_c^1, T_r^1, T_n^1\rangle, \langle W, H, N\rangle)$
11: **return** Acc_{data}
12: **procedure** GETTILEACC(Addr$_{tile}$,$\langle T_c, T_r, T_n\rangle$,$\langle W, H, N\rangle$)
13: $Acc_{tile} \leftarrow 0$
14: **if** $T_c = W$ **and** $T_r = H$ **then**
15: $contSz \leftarrow T_c \cdot T_r \cdot T_n \cdot DW$
16: $Acc_{tile} \leftarrow \mathbb{B}(Addr_{tile}, contSz, BW)$
17: **else**
18: **for** $f \leftarrow 1$ **to** T_n **do**
19: Addr$_f \leftarrow Addr_{tile} + (f-1) \cdot H \cdot W$
20: **if** $T_c = W$ **then**
21: $contSz \leftarrow T_c \cdot T_r \cdot DW$
22: $Acc_{tile} \leftarrow Acc_{tile} + \mathbb{B}(Addr_f, contSz, BW)$
23: **else**
24: $contSz \leftarrow T_c \cdot DW$
25: **for** $r \leftarrow 1$ **to** T_r **do**
26: $Addr_{(f,r)} \leftarrow Addr_f + (r-1) \cdot W$
27: $Acc_{tile} \leftarrow Acc_{tile} + \mathbb{B}(Addr_{(f,r)}, contSz, BW)$
28: **return** Acc_{tile}
29: **procedure** GETTILEDIM$(c, r, n, \langle T_c, T_r, T_n\rangle, \langle W, H, N\rangle)$
30: $T_c^1 \leftarrow min(T_c, W - c)$
31: $T_r^1 \leftarrow min(T_r, H - r)$
32: $T_n^1 \leftarrow min(T_n, N - n)$
33: **return** $\langle T_c^1, T_r^1, T_n^1\rangle$

define inputs and outputs of the algorithm

Algorithm 1 computes the number of bytes accessed from off-chip memory for the 3D data. Each data element is represented by DW bytes. The algorithm takes the address of the data ($Addr_{data}$), tile dimensions, data shape, and δ as input and iterates lines 4–10 for all the tiles. It computes indices of the first element of the tile in the 3D array of the data using the tile index (x, y, z) and δ in lines 5–7. Using the indices of the first element it computes the address of the tile in line 8, and dimensions of the tile in line 9. The algorithm computes the total number of

bytes accessed from off-chip memory by accumulating the number of bytes accessed for each tile in line 10. The address of the tile in line 8 is computed by the function GetTileAddr using equation 2.5 below

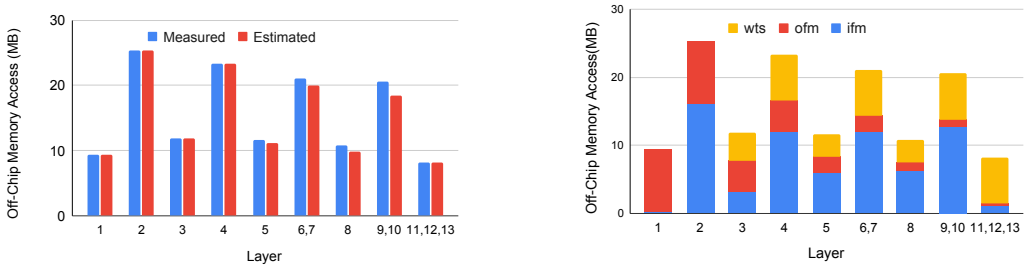
$$GetTileAddr(Addr_{data}, c, r, n, \langle W, H, N\rangle) = Addr_{data} + c + r \cdot W + n \cdot W \cdot H \tag{2.5}$$

If the data dimensions are not multiples of corresponding tile dimensions, the tiles at the border of the data (e.g., the right-most or bottom-most tiles) will have smaller dimensions. The function GetTileDim computes the dimensions of the tiles.

The function GetTileAcc in Lines 12–27 computes the addresses and number of bytes ($contSz$) for each off-chip memory transaction. $contSz$ is the number of contiguous bytes to access in a transaction. Let T_n is the number of frames in the tile. If the tile width (T_c) is the same as the data width (W) (line 20), then all the data elements of T_r rows of a frame are contiguous, and the complete tile can be accessed using T_n number of addresses. If the condition in line 14 is true, then all elements of the tile are contiguous and can be accessed using a single transaction of $T_c \cdot T_r \cdot T_n \cdot DW$ bytes (line 15). In all other cases, the tile is accessed using row addresses computed in line 26. The function computes the number of bytes accessed from off-chip memory for a tile (Acc_{tile}) in lines 16, 22, and 27 using equation 2.3.

DNN accelerators access layer data (ifm, ofm, and weights) partitioned into tiles. The tile dimensions should be chosen carefully to minimize the transfer of extra bytes to reduce off-chip memory access. We have proposed a bus width aware approach (BWA) that factors in the architectural parameters to precisely compute the off-chip memory access of 3D tiles and the layer data.

Figure 2.8a shows the comparison between the off-chip memory accesses estimated by the analytical framework and measurements performed on Xilinx FPGA for different layers of varying shapes and sizes of a popular CNN, VGG16. The experimental results on popular CNNs, AlexNet, and VGG16, show that the difference between estimated and measured off-chip memory accesses is less than 4%. The framework is also helpful in analyzing the layer-wise distribution and breakdown of memory accesses, as shown in Figure 2.8b. The proposed framework is a valuable tool for quickly analyzing the memory accesses and data access energy for different tile dimensions, and data reuse schemes to search for the optimal solution.



(a) (b)

Figure 2.8: Off-chip memory accesss of VGG16 layers (a) Comparison between the analytical framework and measured on hardware. (b) Breakdown by data type using analytical framework.