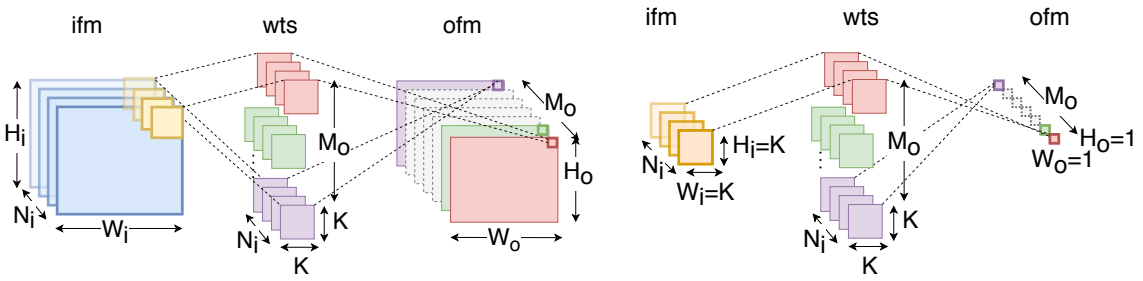
Chapter 3

Optimizing the Performance of CNN Accelerators

CNNs are the state of the art machine learning algorithms. CNNs can achieve human-like accuracy in computer vision-related tasks. In order to achieve high accuracy, modern CNNs use deep hierarchy of layers and perform compute-intensive and memory-intensive operations. CNN accelerators use many processing elements to exploit parallelism to speed up the computations. However, limited off-chip memory bandwidth limits their performance. In addition, considerable data transfer volume from the off-chip memory also results in high energy consumption.

CNNs have a sequence of mainly three types of layers: convolution layer (CL), pooling layer, and fully connected layer (FCL). There are several CLs, and a pooling layer usually follows each CL. The last few layers of the CNNs are FCLs. VGG16 has thirteen CLs and last three layers are FC. Similarly, AlexNet has five CLs, followed by 3 FCLs. Each CL and FCL layer takes 3D input frames (*ifm*) and applies filter weights (*wts*) to compute output frames (*ofm*). Pooling layer computations involve sliding a two dimensional filter window over a single channel of *ifm* and selecting one activation from the window using an operation like maximum or average. There are no parameters in pooling layers. Pooling layer helps in reducing the activation sizes and thus the number of parameters in subsequent layers. CL and FC layer computations involve several parameters. The computations of a CL and an FCL are illustrated in Figure 3.1a and 3.1b, respectively.

CNN accelerators have limited on-chip memory size. Layer activations and parameters sizes are too large to fit into the on-chip memory. To peform the computations CNN accelerators apply loop tiling to partition the layer data into small tiles that fit into on-chip memory. Loop tiling is a compiler technique [1] that partitions the loop iteration space and



(a) Convolution Layer (b) Fully Connected Layer

Figure 3.1: Convolution and fully connected layers

large arrays into smaller tiles to increase the data locality and ensures that data fits into smaller memories. Fig. 3.2 shows a layer's data stored in off-chip memory and its tiles in the accelerator's on-chip buffer.

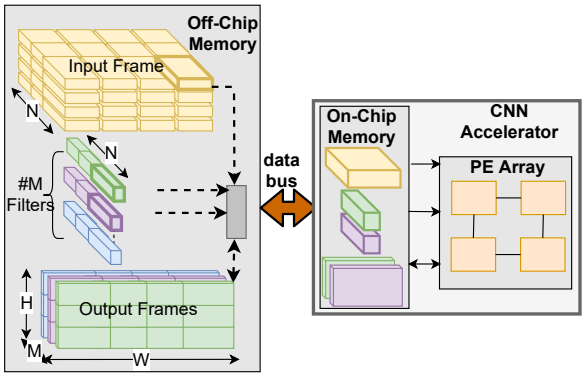


Figure 3.2: CNN layer tiles in off-chip and on-chip memory

Pseudo code of tiled version of a convolution layer is shown in Listing 3.1. The outer 5 loops labelled as L_D, L_H, L_W, L_M, L_N selects the tiles of the data and inner loops perform computations on the selected tiles. Order of the outer loops decide the sequence in which different tiles of data are processed. Each of the $5!$ permutations of the outer loop results in a valid schedule. There are $4!$ permutations in which loop L_M appears as the innermost loop. In these permutations, all iterations of innermost loop L_M uses the same *ifm* data. In all such loop orderings, load statement of *ifm* tile can be moved before the innermost loop L_M and reused in all its iterations [49]. The movement of ifm tile loading, reduces the number of number of off-chip accesses of ifm tile by a factor of $\frac{M}{T_m}$. This ordering scheme exploits the reuse of ifm data and referred as input reuse oriented scheme (IRO). Similarly, the set of $4!$ permutations in which loop L_N is an innermost loop, exploits the data reuse of *ofm* and referred to as output reuse oriented (ORO). The remaining $3 \times 4!$ permutations in which loops L_D, L_R, L_C appears as the inner loops, exploits the weight data reuse and referred to as Weight Reuse Oriented

exploit

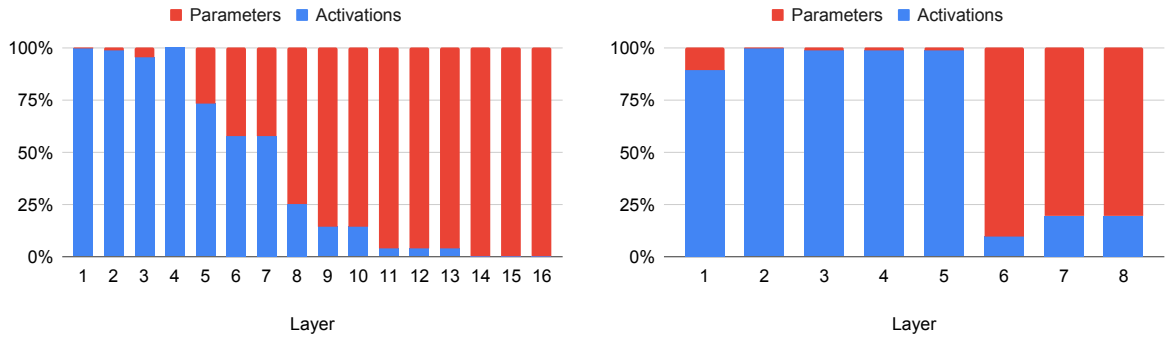
```
1  L_D: for(d=0;d<D;d++) {
2    L_H:  for(row=0;row<H;row+=Tr) {
3      L_W:   for(col=0;col<W;col+=Tc) {
4        L_N:   for(ti=0;ti<N;ti+=Tn) {
5          L_M:   for(to=0;to<M;to+=Tm) {
6            //load wts tile
7            //load ifm tile
8            //load ofm tile
9            for(trr=row;trr<min(row+Tr,H);trr++) {
10             for(tcc=col;tcc<min(col+Tc,W);tcc++) {
11               for(too=to;too<min(to+Tm,M);too++) {
12                 for(tii=ti;tii<min(ti+Tn,N);tii++) {
13                   for(i=0;i<K;i++) {
14                     for(j=0;j<K;j++) {
15                       ofm[d][to][row][col] += weights[to][ti][i][j] * ifm
   [d][ti][S*row+i][S*col+j];
16           } } } } } }
17           //store ofm tile}
18 } } } }}
```

line 15: better to have a line break after * and maintain indentation

Listing 3.1: Pseudo code of a tiled convolution layer

scheme(WRO). The amount of data reuse in different data reuse scheme varies with layer shape.

CNN's layers have varying shapes. Fig. 3.3 shows the parameters and activation proportions in convolution (CL) and fully connected layers (FC) of VGG16 and AlexNet. First few layers have large volume of activations (*ifm* and *ofm*) and last few CLs and FCLs have large volume of parameters(*wts* and *biases*). Scheduling schemes which optimizes the off-chip memory acccesses of activations will work well in first few layers but may not work well for deeper layers, which have large volume of *wts*. The scheduling scheme optimal for one layer may be suboptimal for other layers. Therefore, each layer need to be analyzed here.



(a) VGG16 (b) AlexNet

Figure 3.3: Parameters and activations proportions in CL and FC layers of CNNs.

3.1 Related Work

Zhang et al. [49] used loop tiling to optimize the off-chip memory accesses. They expressed the off-chip memory access as a function of tile dimensions and layer shape and determined optimal tile dimensions by enumerating all the legal tile dimensions. To reduce the hardware design complexity, they determined a global optimal tile dimension and used a common data reuse scheme for all the layers. Due to varying layer shapes, the optimal tile dimension and data-reuse scheme for different layers vary. Li et al. [28] proposed a layer-wise adaptive data partitioning and scheduling scheme to overcome this. However, their approach ignored the architectural parameters and address alignment and assumed that all tiles of the same dimensions have the same off-chip memory accesses. With this assumption, the tile dimensions determined by their approaches are suboptimal.

3.2 Off-Chip Memory Accesses of CNN Layers

3.2.1 Off-chip memory access of CL

Where is the linkage between the equations of chapter 2 and the equations derived here?

A CNN accelerator accesses 3D data of *ifm, ofm*, and *wts* of each layer partitioned into tiles. It accesses tiles of the layer from off-chip memory one or multiple times. Trip counts of the tiles depend on the layer shape, tile dimensions, and the data reuse scheme. If the batch size is D, layer shape is $\langle W_o, H_o, N_i, M_o\rangle$ and tiling parameters are $\langle T_{c_o}, T_{r_o}, T_{n_i}, T_{m_o}\rangle$, trip counts of tiles in IRO, ORO, and WRO schemes can be expressed as the rows of the matrix $\mathbf{R}$ in Equation 3.1, where columns represent *ifm, ofm*, and *wts* trip counts.

how do you derive this?

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_{iro} \\ \mathbf{r}_{oro} \\ \mathbf{r}_{wro} \end{bmatrix} = \begin{bmatrix} D & (2\lceil\frac{N_i}{T_{n_i}}\rceil - 1)D & \lceil\frac{H_o}{T_{r_o}}\rceil\lceil\frac{W_o}{T_{c_o}}\rceil D \\ \lceil\frac{M_o}{T_{m_o}}\rceil D & D & \lceil\frac{H_o}{T_{r_o}}\rceil\lceil\frac{W_o}{T_{c_o}}\rceil D \\ \lceil\frac{M_o}{T_{m_o}}\rceil D & (2\lceil\frac{N_i}{T_{n_i}}\rceil - 1)D & 1 \end{bmatrix} \tag{3.1}$$

First we compute the off-chip memory accesses for one trip of each data type using Algorithm 1 as below

do you mean algorithm 1 of chapter 2?

$$\begin{aligned} \mathbb{B}_{ifm} &= \textit{BWA}(Addr_{ifm}, \langle T_{c_i}, T_{r_i}, T_{n_i}\rangle, \langle W_i, H_i, N_i\rangle, \delta) \\ \mathbb{B}_{ofm} &= \textit{BWA}(Addr_{ofm}, \langle T_{c_o}, T_{r_o}, T_{m_o}\rangle, \langle W_o, H_o, M_o\rangle, 0) \\ \mathbb{B}_{wts} &= M_o \cdot \textit{BWA}(Addr_{wts}, \langle K, K, T_{n_i}\rangle, \langle K, K, N_i\rangle, 0) \end{aligned}$$

where $\delta = (K - S)$ is the number of overlapping elements between adjacent ifm tiles, K is the filter size, and S is the stride. $\mathbb{B}_{ifm}$, $\mathbb{B}_{ofm}$, and $\mathbb{B}_{wts}$ are number of bytes accessed from off-chip memory for one trip of *ifm, ofm* and *wts* respectively. $\langle W_i, H_i, N_i \rangle$ are the *ifm* data shape and $\langle T_{c_i}, T_{r_i}, T_{n_i} \rangle$ are the *ifm* tile dimensions, which are related to *ofm* layer shape and tile dimensions as below

how do you derive this?

$$H_o = (\frac{H_i + 2 \cdot P - K}{S} + 1),\ W_o = (\frac{W_i + 2 \cdot P - K}{S} + 1)$$
$$T_{r_o} = \frac{T_{r_i} - K}{S} + 1,\ T_{c_o} = \frac{T_{c_i} - K}{S} + 1 \tag{3.2}$$

where P is the padding. The total number of bytes accessed for jth reuse scheme ($\mathbb{B}_j$) can be expressed as following sum

$$\mathbb{B}_j = \mathbf{r_j} \cdot \begin{bmatrix} \mathbb{B}_{ifm} & \mathbb{B}_{ofm} & \mathbb{B}_{wts} \end{bmatrix}^T \tag{3.3}$$

where $\mathbf{r}_j$ is the row vector of the matrix $\mathbf{R}$ for the jth scheme.

3.2.2 Optimization problem

Now, we present determining the optimal tile dimensions as a constraint optimization problem. Tiles of *ifm, ofm* and *wts* reside in on-chip memory. Volume of the tiles is given by equation Equation 3.4 below

$$\begin{bmatrix} V_i \\ V_o \\ V_w \end{bmatrix} = \begin{bmatrix} T_{c_i} \cdot T_{r_i} \cdot T_{n_i} \\ T_{c_o} \cdot T_{r_o} \cdot T_{m_o} \\ K^2 \cdot T_{n_i} \cdot T_{m_o} \end{bmatrix} \tag{3.4}$$

where V_i, V_o, and V_w are the sizes of *ifm, ofm*, and *wts* tiles, respectively. If the on-chip memory buffer size is $buffSize$ and each data element is represented by DW bytes, then constraints on tile dimensions are

$$\begin{aligned} &(V_i + V_w + V_o) \cdot DW \leq buffSize \\ &0 < T_{c_o} \leq W_o,\ \ 0 < T_{r_o} \leq H_o \\ &0 < T_{n_i} \leq N_i,\ \ 0 < T_{m_o} \leq M_o \end{aligned} \tag{3.5}$$

Determining the tile dimensions which minimize the off-chip memory accesses, expressed by equation Equation 3.3, is a constraint optimization problem. The number of bytes accessed from off-chip memory using jth reuse scheme $\mathbb{B}_j$ (Equation 3.3) and the constraints (Equation 3.5)

are non-linear functions of four variables $\langle T_{c_o}, T_{r_o}, T_{n_i}, T_{m_o} \rangle$, and thus solving it is non-trivial.

3.2.3 Off-chip memory access of FCL

The computations of FCLs are special case of CLs with additional constraints on layer shapes and parameters. The *ifm* volume is same as a filter volume i.e., $H_i{=}W_i{=}K$, padding $P{=}0$, and stride $S{=}1$. In CNNs, typical values of K are 1, 3, 5, 7 and 11. Due to small values of H_i and W_i, these dimensions are not partitioned ($T_{r_i}{=}T_{c_i}{=}K$). *ofm* layer shape and tile dimensions computed using Equation 3.2 are $\langle 1, 1, M_o \rangle$ and $\langle 1, 1, T_{m_o} \rangle$.

For FCLs, trips count of different data reuse schemes can be computed using equation 3.1and the number of bytes accessed from off chip memory ($\mathbb{B}^{FC}$) using Equation 3.3 by applying the layer shape constraints. The constraints on tile dimensions are given by Equation 3.5. T_{n_i} and T_{m_o} are the unknowns to be determined in Equation 3.3. Determining the optimal tile dimensions for FCLs is a constraint optimization problem, similar to CL.

3.3 Implementation and Results

3.3.1 Implementation

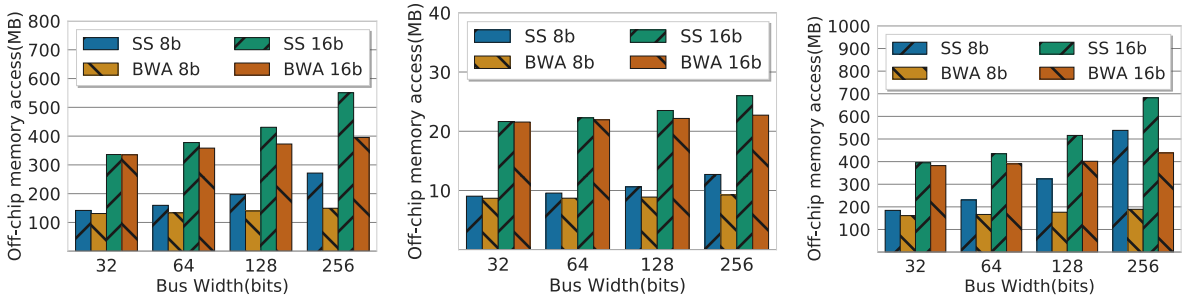
The optimal tile dimensions can be determined by computing the number of bytes accessed from off-chip memory ($\mathbb{B}$) at all the feasible points in the solution space. We have developed the model that computes $\mathbb{B}$ of the CLs using the BWA approach (Algorithm 1) and finds the optimal tile dimensions. The tool also analyses the $\mathbb{B}$ for different data reuse schemes to suggest the best scheme for each CL. It can be configured for different on-chip memory sizes, bus widths, and data bit width. It took less than 60 minutes to determine the optimal solution for VGG16 on Intel Core i7-6700 CPU (@3.40GHz×8).

3.3.2 Validation

hardware implementation of …. in Xilinx FPGA

what is this?

We have validated the number of bytes accessed from off-chip memory computed by BWA (Algorithm 1) using the Xilinx tools. Our validation code is implemented using the Xilinx SDSoC framework, SDx v2018.3, which generates hardware functions from high-level languages like C/C++. We used the SDx pragmas to use zero_copy as a data mover. The Xilinx tools provide the option to integrate the AXI Performance Monitor (APM) IP [46], which captures the real-time performance metrics like bus latency, amount of memory traffic for connected AXI interfaces. The target platform is ZedBoard, working at 100MHz frequency



(a) $\mathbb{B}_{VGG16}$ (b) $\mathbb{B}_{AlexNet}$ (c) $\mathbb{B}_{ResNet}$

Figure 3.4: Off-chip memory access of convolution layers for 8 and 16 bits data width. BWA: Bus Width Aware, SS: SmartShuttle

and off-chip memory (DRAM) is accessed using 64 bits AXI bus. Our validation code takes the 3D shape, and tile dimensions as input and the generated hardware functions access the 3D data from DRAM using loop tiling. The integrated APM IP logs the number of bytes and latency of off-chip memory access transactions using which we validated the number of bytes accessed from off-chip memory computed by our approach for different 3D data shapes and tile dimensions.

3.3.3 Benchmarks

We carried out experiments on three popular CNN networks, VGG16 [38], AlexNet [26], and ResNet [22] having 8, 16, and 50 layers, respectively. These CNNs have varying shapes and use filters of dimensions 1×1, 3×3, 5×5, 7×7, and 11×11. To compare the results with other approaches, we have used the on-chip buffer size of 108 KB, batch size of 3 for VGG16, and 4 for ResNet and AlexNet, as used by Eyeriss [9] and SmartShuttle [28].

3.3.4 Baselines

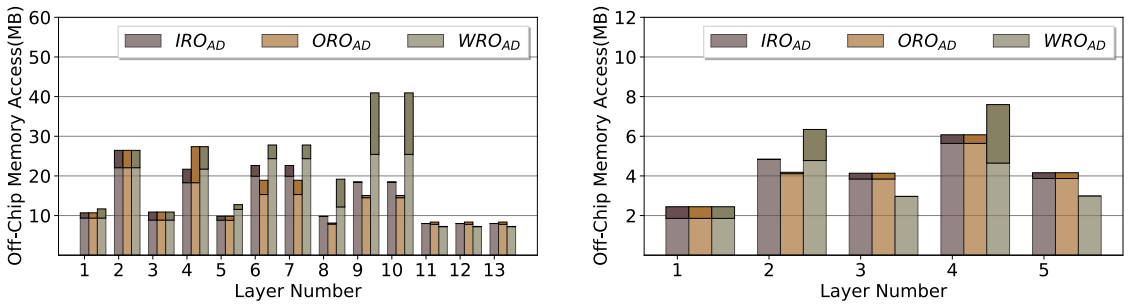
We have implemented the SmartShuttle (SS) [28] approach to compare the results with our bus width aware (BWA) approach. SS statically determines the partitioning and scheduling scheme for each layer. It performs better than strategies that use a fixed tile dimension and data reuse scheme for all layers, e.g., Eyeriss [9] and Zhang [49].

3.3.5 Results

3.3.5.1 Impact of Bus Width on memory access of CLs

Fig. 3.4 shows the number of bytes accessed from off-chip memory ($\mathbb{B}$) of CLs of the CNNs for different bus widths for 8 and 16 bits data width and 108 KB on-chip buffer size. For a wider data bus, $\mathbb{B}$ is more. BWA approach reduces the impact of bus-width on off-chip memory accesses, as it selects the tile dimensions considering the bus width and address alignments. Whereas, tile dimensions selected by SS remains the same, irrespective of bus width of the accelerator, which results in a large value of $\mathbb{B}$. As shown in Fig. 3.4, BWA reduces $\mathbb{B}$ compared to SS for the three CNNs. For ResNet:50 it reduces $\mathbb{B}_{ResNet}$ by 13%, 28%, 46%, and 65% for 8 bits data width and by 10%, 22% and 36% for 16 bits data width on 64, 128, and 256 bits wide data bus, respectively, compared to SS. BWA reduces $\mathbb{B}_{VGG16}$ by 8%, 16%, 29%, and 45% and $\mathbb{B}_{AlexNet}$ by 4%, 9%, 16% and 27% on 32, 64, 128, and 256 bits wide buses respectively, for 8 bits data width. The impact of bus width is significant when accessing low-resolution data on a wide data bus. For 16 bits data width, the effectiveness of the BWA approach is noticeable for 64 or wider data buses. For 16 bits data width, BWA reduces $\mathbb{B}_{VGG16}$ by 5%, 13.5% and 28% and $\mathbb{B}_{AlexNet}$ by 1.5%, 5.7% and 13% compared to SS on 64, 128, and 256 bits wide data bus, respectively.

3.3.5.2 Off-Chip Memory Access of Data Reuse Schemes



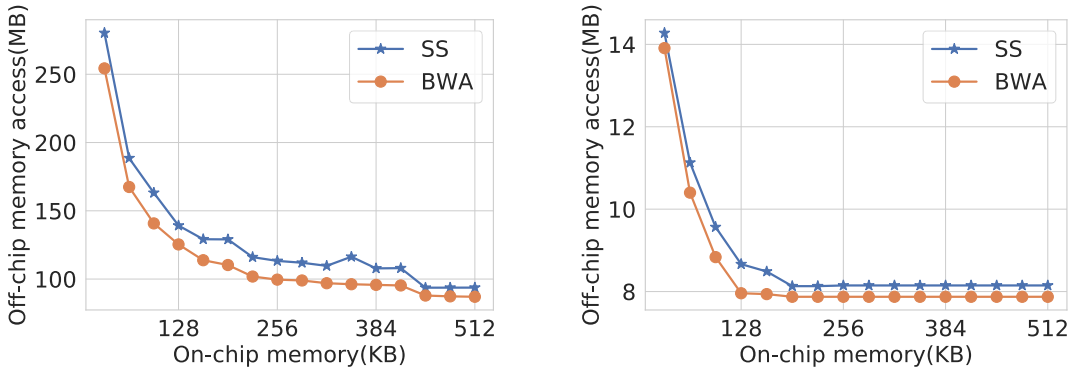
(a) VGG16 CLs (b) AlexNet CLs

Figure 3.5: Layer wise off-chip memory access for IRO, ORO and WRO schemes.

Figure 3.5 shows the layer-wise off-chip memory access of CLs of VGG16 and AlexNet for the three data reuse schemes using 64 bits wide bus and 8 bits data width. The results show that a single data reuse scheme is not optimal for all the layers. In the first few layers, IRO and ORO perform better than the WRO scheme, while in the last few layers, WRO outperforms the other two schemes. The solid color at the top of the bars shows the reduction in off-chip memory accesses when optimal tile dimensions are selected using the BWA approach compared

to when tile dimensions are selected using the tile size-based approach. The BWA approach performs better than the tile size based approach for all the three data reuse schemes.

3.3.5.3 On-chip Buffer Size



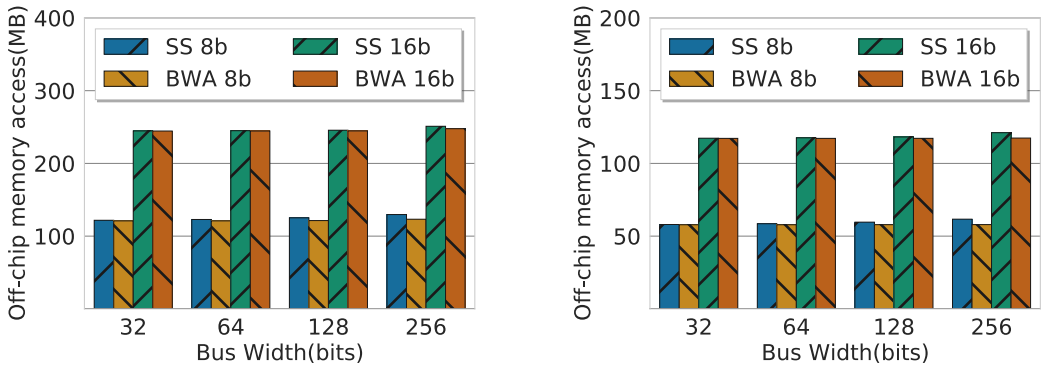
(a) $\mathbb{B}_{VGG16}$ (b) $\mathbb{B}_{AlexNet}$

Figure 3.6: Off-chip memory access for varying on-chip buffer sizes. BWA: Bus Width Aware, SS:SmartShuttle

Fig. 3.6a and Fig. 3.6b compares the number of bytes accessed from off-chip memory ($\mathbb{B}$) for different on-chip buffer sizes ($buffSize$) for VGG16 and AlexNet, respectively. Large on-chip buffer can accommodate larger tiles, which reduces the tiles trip counts (Equation 3.1) and therefore the total off-chip memory accesses of the CNN (Equation 3.3). This behavior is observed for both the CNNs in Fig. 3.6a and Fig. 3.6b. Both the approaches select the dimensions of the tiles with the constraints of on-chip buffer size. However, the proposed BWA approach performs better as it considers the address alignments and bus width to reduce unwanted data transfers to optimizes the off-chip memory accesses, while SS approach ignores the architectural parameters.

3.3.5.4 Impact of Bus Width on $\mathbb{B}$ of FCLs

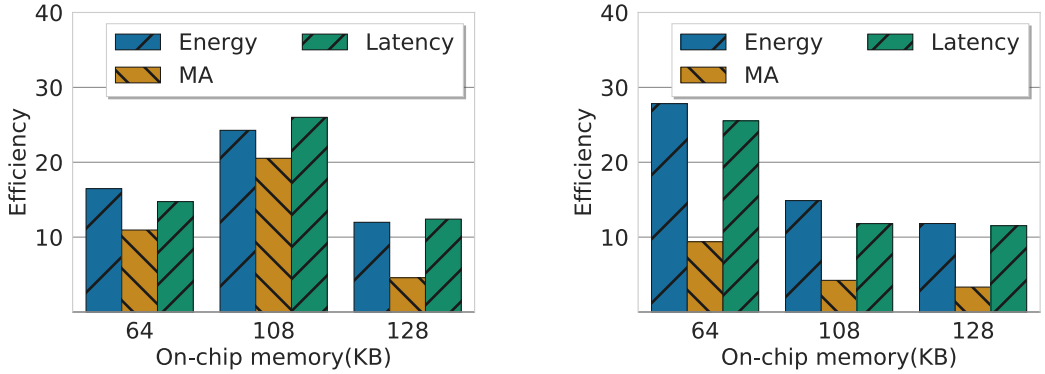
In FCLs, the height and width of tiles and data are the same. Tiles in FCL can be accessed from off-chip memory using a single transaction which results in fewer transactions in FCLs compared to CLs. Whereas in CLs, multiple transactions are required to access different rows of the tiles. Thus the impact of bus width is less on $\mathbb{B}$ of FCLs as compared to CLs. Figure 3.7a and Figure 3.7b shows the off-chip memory accesses of FCLs of VGG16 and AlexNet, respectively, for 8 bits data width. BWA reduces $\mathbb{B}^{FC}_{VGG16}$ by 1%, 2%, 3%, and 4% and $\mathbb{B}^{FC}_{AlexNet}$ by 0.2%, 1%, 3% and 6% on 32, 64, 128, and 256 bits wide buses, respectively.



(a) VGG16 (b) AlexNet

Figure 3.7: Off-chip memory access of Fully connected layers. BWA: Bus Width Aware, SS:SmartShuttle

3.3.5.5 Latency And Energy Analysis



(a) VGG16 (b) AlexNet

Figure 3.8: Energy and latency efficiency. BWA: Bus Width Aware, SS:SmartShuttle

Using our FPGA implementation, we measured the memory access latencies and execution time for CLs for the optimal tile dimensions determined using the approach described in 3.2. To estimate the energy efficiency achieved by the BWA compared to the SS approach, we computed the energy consumption using the following equation [**?**]

$$E = P \cdot Time + \mathbb{B} \cdot E_{DDR} \tag{3.6}$$

where P is the FPGA design power reported by the Vivado synthesis tool, $Time$ is the execution time, $\mathbb{B}$ is the number of bytes accessed from off-chip memory logged using Xilinx APM IP, and E_{DDR} is the off-chip memory access energy per bit. We have used E_{DDR}=70 pJ/bit, a typical value for the DDR3 memory access energy [**?**].

Figure 3.8a and Figure 3.8b show the energy, off-chip memory accesses, and latency

efficiency achieved using the BWA compared to the SS approach for VGG16 and AlexNet, respectively, for 8 bits data width and 64 bits bus width. We observed that the changes in energy and latency are proportional to the changes in memory access. This observation confirms that off-chip memory access dominates the energy consumption of the CNN accelerators.

3.4 Summary

Off-chip memory accesses dominate the energy consumption of CNN accelerators. Loop tiling is a common technique to partition the layer data into smaller tiles that fit into on-chip memory. The tile dimensions have a significant impact on the off-chip memory accesses of these accelerators. In this work, we propose a bus width and address alignment aware approach to compute the off-chip memory accesses of 3D data. Our tool statically analyses the memory accesses to find the optimal tile dimensions for CNN accelerators. Experimental results show that our approach reduces off-chip memory accesses of the CLs of VGG16 by 16%, 29%, and of AlexNet by 9%, 16% on 64, and 128 bits data bus respectively for 8 bits data width, compared to the state of the art approach.