On the Impact of Partial Sums on Interconnect Bandwidth and Memory Accesses in a DNN Accelerator

Mahesh Chandra
NXP Semiconductors, India

Abstract— Dedicated accelerators are being designed to address the huge resource requirement of the deep neural network (DNN) applications. The power, performance and area (PPA) constraints limit the number of MACs available in these accelerators. The convolution layers which require huge number of MACs are often partitioned into multiple iterative sub-tasks. This puts huge pressure on the available system resources such as interconnect and memory bandwidth. The optimal partitioning of the feature maps for these sub-tasks can reduce the bandwidth requirement substantially. Some accelerators avoid off-chip or interconnect transfers by implementing local memories; however, the memory accesses are still performed and a reduced bandwidth can help in saving power in such architectures. In this paper, we propose a first order analytical method to partition the feature maps for optimal bandwidth and evaluate the impact of such partitioning on the bandwidth. This bandwidth can be saved by designing an active memory controller which can perform basic arithmetic operations. It is shown that the optimal partitioning and active memory controller can achieve up to 40% bandwidth reduction.

Keywords—DNN, Interconnect bandwidth, optimization, active memory controller, architecture

I. INTRODUCTION

Deep neural networks are used for the state of the art results in multiple applications. The convolution layers used in these networks require huge computing power and data movement. To achieve real time performance, dedicated accelerator architectures are being explored and designed [1,2,3]. However, due to cost, technology and power constraints, number of MACs are still limited in these accelerators. Often, these are not enough to compute all the output feature maps or process all the input feature maps for the most of the layers in state of the art DNNs. So, there is a requirement of partitioning the convolution task into multiple sub tasks. Due to this, only some of the input and output maps are processed in one iteration generating only partially sums.

Since, the CNNs come with different flavors, effective PE utilization is one of the key issues with the DNN accelerators. PE utilization is affected by data flow and partitioning of convolution tasks. Loop optimization techniques such as loop unrolling, loop tiling and loop interchange are used to find the right configuration for the efficient data flow and minimum data movement[4]. Different data reuse strategy are adopted to optimally use the system resources and architect the accelerators [5].

The strategies used for reusing the weights, input activations or output activations improve the PE utilization and reduce the amount of data transfer. However, each of these strategy has a different impact on the data bandwidth (BW) requirement due to the input maps, output maps and weights. These strategy also often require the partial sums to be generated as all channels cannot be accommodated in PEs. These partial sums significantly increase the required bandwidth because the partial sums must be read before being updated. Here, note that in case of local memories, the bandwidth will translate into memory accesses. In case of local memories, it’s a big factor for power consumption, so reducing it is as important as the off-chip or interconnect bandwidth. Though, there is general agreement on the increased bandwidth requirement due to partial sums, a quantitative analysis is missing. Such analysis for multiple CNNs is reported in the section IV of this paper.

The CNNs are feed forward neural networks. Activation of a layer can be computed only when the activations of previous layer or layers are available. So, once a loop tiling approach is decided, we can assume that all the activation for that particular layer will be computed before the next one. Since, there are only limited number of resources available in an accelerator; the neural network model must be partitioned into multiple sub-tasks for execution. Optimally partitioning a tensor is a complex problem and is an active area of research. Thus, the problem of assigning the available hardware resources (compute, memory, bandwidth etc.) to meet the application requirements (accuracy, throughput, power etc.) explodes exponentially. Some system uses fixed assignment while others have used heuristics to assign the resources. Reducing off-chip bandwidth is also used as one of the key criteria for PE allocation [6]. Recently, Kao et al. [7] have applied reinforcement learning for the resource assignment.

The optimum PE utilization also requires the PEs to be assigned according to the layer characteristics [8]. So, the PE allocation for the input and output channels would vary from layer to layer. So far, the PE utilization and off-chip communication has been the main focus for partitioning schemes and on-chip memory bandwidth requirement has not received enough attention. However, as the partial sums require lot of bandwidth, attention must also be paid to BW while partitioning the channels. The bandwidth requirement varies with the partitioning of the input and output feature maps. An optimal partitioning is thus key to the bandwidth requirement in an accelerator. A simple partitioning method is discussed in section II of this paper which considers the PE utilization and bandwidth. This method reduces the bandwidth requirement significantly over other simple methods.

The DNNs have huge requirement of both compute power and data transportation. Researchers are exploring the processing near memory and processing in memory [9] architectures to address these challenges. Processor in memory, intelligent RAM and other such architectures integrate the processors within the memory itself. These architectures will have huge impact on the future DNN accelerators. However, for the current set of accelerators, some simple solution such as active memory controller [10] can have huge impact on improving the PPA. This paper
discusses the impact of the active memory controller on the bandwidth. Since, the off-chip data transfers result in high power consumption; the accelerators are designed with large on-chip memory. The active memory controller, discussed here, can also be helpful in optimizing bandwidth in such accelerators and hence, reduce the power consumption.

II. METHOD OF PARTITIONING THE FEATURE MAPS

Simplified code (without considering the stride and different output dimensions) for a convolution layer in a CNN, processing M input feature maps of size \(W_x \times H_x\) and generating \(N\) output maps of size \(W_o \times H_o\), using a kernel \(K \times K\), can be written as:

\[
\begin{align*}
\text{for } (x=0:x<W_x;&) \\
\text{for } (y=0:y<H_y;&) \\
\text{for } (k=(K-1)/2:o<=(K-1)/2;&) \\
\text{for } (l=(K-1)/2:o<=(K-1)/2;&) \\
\text{\_out}(co)[x][y] &= \text{\_in}(co)[x+k][y+l] \times \text{\_wt}(co)[c][k][l];
\end{align*}
\]

For maximum data reuse, we’d ideally like to process all the input map only once and compute the output map at one iteration as more often they are the biggest contributors to the data bandwidth. However, this requires a large number of MACs in the accelerator which is not feasible. In practice, we have limited number of MAC units in the accelerator and so, the input and output maps are partitioned and only partial sums are computed. It means that computing an output feature map requires multiple iteration. Similarly, an input map may be read more than once for computing different output maps. If we process \(n\) input maps and \(m\) output maps in one iteration, the code sequence above is re-written as:

\[
\begin{align*}
\text{for } (co_base=0:co_base<N,co_base=co+n) \\
\text{for } (x=0:x<W_x;&) \\
\text{for } (y=0:y<H_y;&) \\
\text{for } (k=(K-1)/2:o<=(K-1)/2;&) \\
\text{for } (l=(K-1)/2:o<=(K-1)/2;&) \\
\text{\_out}(co_base+\text{co})[co][x][y] &= \text{\_in}(co_base+\text{co})[x+k][y+l] \times \text{\_wt}(co_base+\text{co})[c][k][l];
\end{align*}
\]

The choice of \(m\) and \(n\) trades-offs the data reuse at input and output. In this section, we discuss the first order method for optimal selection of \(m\) and \(n\).

Let’s first make some assumptions for the first order model. Let the number of MACs in the accelerator be \(P\). So, these \(P\) multipliers, that the accelerator can handle, need to be divided among the input and output maps for computing partial sums. Also, assuming that the kernel is not split, any given choice of \(m\) and \(n\) must satisfy:

\[
K^2 \times m \times n < P \tag{1}
\]

The input map will be read \(N/n\) times and output map (partial sums) is written \(M/m\) times. Note that the output map will also have to be read before being updated for all the iteration except the first one. Though, in case of fixed point representation, the partial sums will be larger than the activations and the bandwidth requirement will be even more; for the sake of simplicity, we assume that the partial sum has the same size as activation in the bandwidth calculations here. The Input and output bandwidths \((B_i\text{ and }B_o\text{ respectively here})\) are given by:

\[
\begin{align*}
B_i &= W_i \times H_i \times M \times \frac{N}{n} \tag{2} \\
B_o &= W_o \times H_o \times N \times (2 \times \frac{M}{m} - 1) \tag{3}
\end{align*}
\]

Total bandwidth (in terms of million activation per inference) is given by:

\[
B = B_i + B_o \tag{4}
\]

For the purpose of this model, let’s assume that we can find \(m\) and \(n\) such that:

\[
K^2 \times m \times n = P \tag{5}
\]

The eq \(4\) can be now written as:

\[
B(m) = W_i \times H_i \times M \times \frac{N}{P} \times K^2 \times m + W_o \times H_o \times N \times (2 \times \frac{M}{m} - 1) \tag{6}
\]

To minimize, \(B(m)\),

\[
\frac{dB}{dm} = 0
\]

Which results in

\[
m = \frac{2W_o \times H_o \times \frac{P}{W_i \times H_i \times K^2}}{1 \times (2 \times \frac{M}{m} - 1)} \tag{7}
\]

Value of \(m\) obtained by \(7\) is a real number and cannot be used directly; so, it must be adapted. For the purpose of this analysis, the value of \(m\) is slightly modified so that it is integer and it is a factor of \(M\) (i.e. total number of input channels in the layer). Once \(m\) is known, \(n\) can be computed from \(5\).

III. REDUCING BW WITH ACTIVE SRAM CONTROLLER

As discussed above, to update the result of convolution, the accelerator must read the previous partial sum before adding new one and writing it to memory. So, each operation, except the first one, requires a read and a write operation. Active SRAM controller have been proposed in past to offload some of the CPU tasks in case of data-heavy processing [10]. Similar to these active controller, if we design a memory controller which is capable of doing simple operations like addition, initialization and compare etc., we can avoid reading of the data from the memory and save lot of memory accesses and data bandwidth. There are two main challenges, one, which operations to offload because the memory controller can become as complex as compute engine if care is not taken and two, how to communicate to memory controller about the type of operation. These two issues are addressed in this section.

The working principle is that instead of transporting data (i.e. past partial sums, in this case) all the way to the compute engine (or MAC units), perform these operations (i.e. the addition) locally in memory controller. MACs compute the
### TABLE I. BANDWIDTH (MILLION ACTIVATIONS/IMAGE) REQUIREMENT DIFFERENT PARTITIONING STRATEGIES AND NUMBER OF MACS (P) IN ACCELERATOR

| CNN     | P=512 | P=2048 | P=16384 |
|---------|-------|--------|---------|
|         | Max Input | Max Output | Equal MACs | This Work | Max Input | Max Output | Equal MACs | This Work | Max Input | Max Output | Equal MACs | This Work |
| AlexNet | 61.9    | 94.2    | 26.2    | 25.1    | 52.2     | 64.6      | 13.0     | 12.6    | 9.2      | 10.9      | 7.3       | 4.3       |
| VGG-16  | 1170.3  | 1938.6  | 494.2   | 442.5   | 909.5    | 1309.3    | 269.3    | 237.2   | 207.1    | 241.1     | 151.0     | 83.5      |
| SqueezeNet | 199.6  | 244.8   | 65.9    | 52.0    | 53.6     | 105.2     | 47.4     | 26.2    | 12.6     | 17.3      | 34.8      | 11.1      |
| GoogleNet | 431.7  | 313.0   | 102.5   | 95.5    | 174.6    | 151.6     | 61.2     | 47.7    | 23.8     | 24.1      | 41.6      | 17.5      |
| ResNet-18 | 281.2  | 315.8   | 96.1    | 88.9    | 205.0    | 191.6     | 50.9     | 46.8    | 35.1     | 31.7      | 26.9      | 16.0      |
| ResNet-50 | 5245.2 | 5770.4  | 1059.2  | 952.6   | 2909.0   | 2830.4    | 608.6    | 479.5   | 929.8    | 682.5     | 330.1     | 168.5     |
| MobileNet | 215.0  | 209.2   | 78.5    | 68.3    | 136.8    | 116.2     | 48.8     | 35.0    | 21.9     | 21.0      | 34.9      | 16.1      |
| MNASNet | 884.4   | 1294.1  | 405.3   | 373.4   | 722.0    | 1030.3    | 213.4    | 183.0   | 500.2    | 516.3     | 101.8     | 66.0      |

### TABLE II. BANDWIDTH (MILLION ACTIVATIONS/IMAGE) REQUIREMENT PASSIVE AND ACTIVE MEMORY CONTROLLER DEPENDING ON NUMBER OF MACS (P) IN THE ACCELERATOR

| CNN     | Passive Memory Controller | Active Memory Controller |
|---------|---------------------------|--------------------------|
|         | 512 MACs | 1024 MACs | 2048 MACs | 4096 MACs | 8192 MACs | 16384 MACs | 512 MACs | 1024 MACs | 2048 MACs | 4096 MACs | 8192 MACs | 16384 MACs |
| AlexNet | 25.07    | 17.54    | 12.56    | 8.89     | 6.52     | 4.32      | 17.89    | 12.62   | 8.77     | 6.38      | 4.55      | 3.51      |
| VGG-16  | 442.49   | 321.79   | 237.25   | 169.43   | 112.14   | 83.54     | 315.33   | 225.44  | 161.67   | 123.36    | 89.97     | 63.67     |
| SqueezeNet | 51.98   | 37.47    | 26.22    | 20.04    | 14.12    | 11.10     | 40.06    | 27.35   | 20.76    | 14.87     | 12.61     | 9.78      |
| GoogleNet | 93.46   | 67.17    | 47.65    | 35.20    | 23.23    | 17.51     | 69.90    | 48.37   | 35.77    | 25.95     | 20.63     | 14.62     |
| ResNet-18 | 88.87   | 63.56    | 46.79    | 32.86    | 22.01    | 16.02     | 63.52    | 45.53   | 32.34    | 24.74     | 17.81     | 12.90     |
| ResNet-50 | 952.60  | 691.13   | 479.50   | 349.75   | 232.82   | 168.46    | 691.98   | 480.49  | 346.77   | 242.90    | 183.09    | 121.93    |
| MobileNet | 68.53   | 46.74    | 35.14    | 25.22    | 21.00    | 16.02     | 50.90    | 39.03   | 27.69    | 22.66     | 17.82     | 15.58     |
| MNASNet | 373.41  | 264.36   | 183.01   | 128.27   | 92.35    | 65.96     | 258.91   | 188.75  | 131.06   | 94.92     | 67.80     | 50.40     |

The second problem is to communicate with the SRAM controller. Note that the DNNs require huge data bandwidth and would require the latest advanced bus interface e.g. AXI4. These advanced interfaces support the sideband signals which are part of the infrastructure and travel through the interconnect. For example, AXI4 has the ‘awuser’ signals[11] which can be used for signaling the command to the SRAM controller. The representative block diagram is shown in the figure 1.

```
Compute Engine (DNN Accelerator)
  AXI RA Ch
  AXI Rd Ch
  AXI Wr Ch
  AXI WA Ch
  AXI WAUSER

Interconnect

Memory Controller
  AXI RA Ch
  AXI Rd Ch
  AXI Wr Ch
  AXI WA Ch
  AXI WAUSER
```

Fig. 1. Communication between compute engine and memory controller using advanced bus protocols

For the purpose of analysis in this paper, only offloading of the addition of partial sums to the memory controller is assumed. The analysis shows significant reduction in bandwidth requirement as discussed in next section. Note here that a write operation is now converted to a read-modify-write operation at the SRAM controller, so throughput at SRAM
remains same. To maintain the required throughput at the SRAM controller interface, multiple options are available such as doubling the data width of SRAM, doubling the number of SRAM cuts or doubling the speed. Typically, the on-chip SRAMs are quite big, so, working with multiple cuts may be easily applied in most accelerators. Since, SRAM controller will be able to meet all the requirements at the interface level; there are no new challenges on scheduling etc. because of this architecture.

IV. RESULTS AND DISCUSSION

First, the minimum bandwidth required for some of the popular CNNs is evaluated. Table III lists the cumulative bandwidth required by the convolution layers of some of the popular CNNs, assuming that the output of convolution layer is written to the memory (i.e. no fused operations across layers) and the data is read and written only once (i.e. we have unlimited compute resources and don’t need to write partial sums). Note that the first assumption also applies to the rest of the discussion in this section. This bandwidth is computed for the inference on a 224x224 color image.

| CNN          | BW (M Activations/inference) |
|--------------|------------------------------|
| AlexNet [12] | 0.823                        |
| VGG-16 [13]  | 0.095                        |
| SqueezeNet [14] | 7.304                     |
| GoogleNet [15] | 7.889                     |
| Resnet-18 [16] | 4.666                     |
| Resnet-50 [16] | 28.349                     |
| MobileNet [17] | 10.273                     |
| MNASNet [18]  | 11.001                       |

In a constrained system, the bandwidth impact is decided by the partitioning of input and output maps. Table I compare the bandwidth requirements for the input and output maps partitioning using four different method for three different MAC numbers. These methods are:

1. Allocate the MACs to maximum number of input maps. This will reduce the number of output iterations. This is reported in 1st columns in table I.
2. Allocate the MACs to maximum number of output maps. This will reduce the number of input iterations. This is reported in 2nd columns in table I.
3. Allocate equal number of MACs to input and output channels. This is reported in 3rd columns in table I.
4. Allocate as per the first order optimum number as presented in this paper in section II. This is reported in 4th columns in table I.

Note that the method presented here results in the least bandwidth requirement. Also note that as number of MACs increases, the bandwidth requirement decreases and with a very large number of MACs, it approaches the minimum bandwidth as given in table III. So, it is obvious that the partitioning strategy is very critical in resource constrained systems such edge devices and low power systems.

If the SRAM controller is updated for performing additions locally as discussed in section III, the bandwidth requirement is further reduced significantly. The absolute numbers are given in table II and percentage reduction in figure 2.

As the number of MACs in accelerators is increased, the gain from active SRAM controller is reduced. However, even with relative large number of MACs (16K) in the device, the performance gain varies between 2-38% which is significant. As expected the gain is significantly higher at 19-42% for more constrained compute systems (i.e. less number of MACs).

Fig. 2. Percentage bandwidth saving with active SRAM controller

From the data presented here, it is obvious that significant bandwidth can be saved using the active memory controller. This method cannot be compared with other such methods as there is no other published study to the best of my knowledge, and this method complements other methods published in literature for optimizing resources. Note here that the cost of supporting this is very small logic in memory controller and handling the sideband signals in the interconnect. This cost is almost negligible. Another obvious advantage of reducing this bandwidth is reduced power consumption.

V. CONCLUSION

In this paper, the impact of the partial sum computation on the memory bandwidth is evaluated. A simple method for partitioning the feature maps is proposed that helps in reducing the bandwidth significantly. The theoretical analysis presented here shows that the active memory controller can further reduce the bandwidth requirement. Its correlation with actual power consumption needs to be validated using actual hardware or model. Moreover, an active SRAM can be designed which can offload the SRAM controller which will further reduce the power consumption. Though, it requires detailed gain vs. cost analysis; once proven, this approach can be useful in designing the low power machine learning accelerators.

REFERENCES

[1] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, “A Survey of Accelerator Architectures for Deep Neural Networks”, Engineering, 2020
[2] J. Wang, J. Lin and Z. Wang, “Efficient Hardware Architectures for Deep Convolutional Neural Network,” IEEE Transactions on Circuits and Systems I: Regular Papers, 65(6), 2018
[3] V. Sze, Y. Chen, J. Emer, A. Suleiman and Z. Zhang, "Hardware for machine learning: Challenges and opportunities", 2017 IEEE Custom Integrated Circuits Conference (CICC), Austin, TX, 2017
[4] Y. Ma, Y. Cao, S. Vrudhula, and J. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural
networks.” In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2017.

[5] Y. Chen, J. Emer and V. Sze, "Using Dataflow to Optimize Energy Efficiency of Deep Neural Network Accelerators", IEEE Micro, 37(3), 2017

[6] J. Li, G. Yan, W. Lu, S. Jiang, S. Gong, J. Wu, and X. Li, "SmartShuttle: Optimizing off-chip memory accesses for deep learning accelerators", In 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018

[7] S.C. Kao, G. Jeong, and T. Krishna, "ConfiaciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning", arXiv preprint arXiv:2009.02010, 2020

[8] B. Liu, X. Chen, Y. Wang, Y. Han, J. Li, H. Xu, and X. Li, "Addressing the issue of processing element under-utilization in general-purpose systolic deep learning accelerators", In Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019

[9] S. Ghose, K. Hsieh, A. Boroumand, and R. Ausavarungnirun, “Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions”, arXiv:1802.00520v1, 2018

[10] J. Yoo, S. Yoo, and K. Choi, “Active Memory Processor for Network-on-Chip-Based Architecture”, IEEE Trans. on computers, 61(5), 2012

[11] ARM, "Axi4-stream protocol specification"

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks” NIPS, 2012

[13] K. Simonyan, and A. Zisserman, “deep convolutional networks for large-scale image recognition”, arXiv:1409.1556, 2014

[14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “Squeezenet: Alexnet-level accuracy with 50x fewer parameters an <0.5MB model size”, arXiv:1602.07360v4, 2016

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions." Proceedings of the IEEE CVPR, 2015

[16] K. He, X. Zhang, S. Ren, And J. Sun, "Deep residual learning for image recognition", Proceedings of the IEEE CVPR, 2016

[17] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks”, In Proceedings of the IEEE CVPR, 2018

[18] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, “Mnasnet: Platform-aware neural architecture search for mobile”, In Proceedings of the IEEE CVPR, 2019