Improving Memory Access Locality for Vectorized Bit-serial Matrix Multiplication in Reconfigurable Computing

Lahiru Rasnayake and Magnus Själander
[firstname.lastname]@ntnu.no
Norwegian University of Science and Technology (NTNU)
Trondheim, Norway

***Abstract*—Low-precision matrix multiplication has gained significant interest in the research community due to its applicability in the quantized neural network domain. As a result, a multitude of variable precision hardware designs have been proposed since fixed-precision hardware causes under-utilization of the hardware resources due to the low and varying precision in such applications. Bit-serial hardware takes advantage of the frugal nature of bit-serial computations that can operate on only as many bits as necessary. A bit-serial matrix multiplication consists of a summation of weighted binary matrix multiplications. In this work, we study the inherent locality of bit-serial matrix multiplications and propose a locality-aware scheduling algorithm that eliminates redundant data fetches from memory. The proposed schedule improves with up to 76% compared to a schedule that computes each binary matrix multiplication in sequence.**

I. Introduction

Matrix-matrix multiplication is a key computational kernel used in many applications [1] of which deep learning inference is one of the most popular. The large number of matrix multiplications involved in deep neural networks make them heavily memory bound in addition to being computationally intensive. Several techniques pertaining to the optimization of matrix multiplication are available that still do not account for the heavy memory requirements in deep learning. Thus, significant effort has gone into memory oriented optimizations in deep neural networks [2] such as weight pruning, which reduce the number of multiplications, and quantization, which reduce the precision of the multiplications.

Despite the general advantage from quantization [3], the varying precision of inputs due to differences in quantization across layers in a neural network [4] means that conventionally designed fixed-precision hardware are under-utilized. In the case of FPGAs, even though the possibility of reconfigurability exists, reconfiguration gives rise to overheads. An alternative to mitigate this under-utilization is the use of bit-serial computations [5] where the integer or fixed-point matrix multiplication is represented as a sum of weighted binary matrix multiplications (see Algorithm 1). The primary issue of bit-serial computations has been the significant latency due to the sequential execution of high-precision data. However, bit-serial computations can be highly beneficial for low-precision matrix multiplication as the latency is low due to the low-precision and is offset by the inherent data-level parallelism in matrix multiplications. Parallelism can be obtained not only through instantiation of multiple bit-serial units but also through vectorization as demonstrated in BISMO [6].

For architectures that are optimized with memory efficiency as a primary goal, it is important to identify memory efficient schedules. Naive scheduling commonly result in loss of locality with redundant memory accesses as a result. We propose a locality-aware scheduling algorithm that ensure high operational intensity for low-precision matrices executed on a vectorized bit-serial accelerator. We use BISMO as a reference platform for bit-serial matrix multiplications to evaluate the potential benefits of the proposed scheduling algorithm.

II. Bit-Serial Matrix Multiplication

Conventional digital arithmetic circuits of today mostly belong to the bit-parallel paradigm where all the bits of a word are computed in parallel, whereas the bit-serial paradigm involves dealing with individual bits of the word in a serialized fashion. The bit-serial paradigm improves full-circuit utilization at reduced circuit size with the main disadvantage of incurring additional latency.

Algorithm 1 represents a straight forward algorithm by which bit-serial matrix multiplication can be performed [6], with Equation 1 representing how a single element of the result matrix is calculated. Priority is given to the calculation of the inner product of two binary matrices (e.g., $L^{[0]} \cdot R^{[0]}$) before proceeding to calculate the inner product of another bit-precision (e.g., $L^{[0]} \cdot R^{[1]}$).

Algorithm 1 Bit-serial matrix multiplication on unsigned integers.

Input: $m \times k$, w-bit matrix L and $k \times n$, a-bit matrix R
Output: $P = L \cdot R$
 # For binary matrices of different precision
for $i \leftarrow 0 \ldots w-1$ **do**
  for $j \leftarrow 0 \ldots a-1$ **do**
    weight $= 2^{i+j}$
    # Perform binary matrix multiplication
    for $r \leftarrow 1 \ldots m$ **do**
      for $c \leftarrow 1 \ldots n$ **do**
        for $d \leftarrow 1 \ldots k$ **do**
          $P_{rc} = P_{rc} + \text{weight} \times (L^{[i]}_{rd} \times R^{[j]}_{dc})$

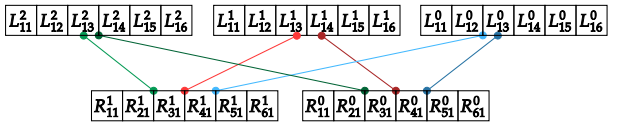
$$\forall\, r, c \mid r \in \{1 \ldots m\},\ c \in \{1 \ldots n\}$$

$$P_{rc} = \sum_{i=0}^{w-1} \sum_{j=0}^{a-1} 2^{i+j} \times \sum_{r=1}^{m} \sum_{c=1}^{n} \sum_{d=1}^{k} (L_{rd}^{[i]} \times R_{dc}^{[j]}) \quad (1)$$

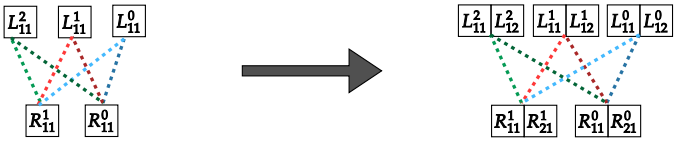
Fig. 1 shows an example of a bit-serial matrix multiplication between a three-bit left-hand (L) matrix and a two-bit right-hand (R) matrix (in its transposed form). For simplicity, each matrix consists of a single row with six columns.

Fig. 1a highlights the access pattern resulting from Algorithm 1. A complete bit-matrix multiplication is computed before proceeding to another bit-precision, i.e., $L^{[0]} \cdot R^{[0]}$ (dark blue) is computed before $L^{[0]} \cdot R^{[1]}$ (light blue) followed by $L^{[1]} \cdot R^{[0]}$ (dark red) and so forth.

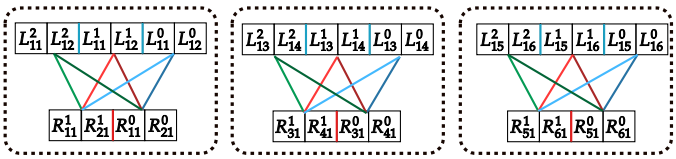
Given a resource-constrained compute engine this schedule causes redundant memory fetches. Assume that only a single bit-matrix of the left-hand ($L^{[i]}$) and right-hand ($R^{[j]}$) matrices could fit in on-chip memory then the right-hand matrices ($R^{[0]}$ and $R^{[1]}$) would be fetched once for each of the left-hand matrices ($L^{[0]}$, $L^{[1]}$, and $L^{[2]}$). Thus, resulting in $R^{[0]}$ and $R^{[1]}$ being fetched three times instead of ideally only a single time.



(a) Access pattern given Algorithm 1.



(b) Locality in bit-serial matrix multiplication.



(c) Locality-aware schedule given Algorithm 2.

Fig. 1: Bit-serial matrix multiplication access patterns.

Fig. 1b (left) shows the inherent dependencies between the bit-precisions in a bit-serial multiplication. If the computation starts with an element from $L^{[0]}$ it should be multiplied with its corresponding element in each right-hand matrix ($R^{[0]}$ and $R^{[1]}$ as highlighted in blue). Each of the right-hand elements in turn should be multiplied with their corresponding element in the remaining left-hand matrices ($L^{[1]}$ highlighted in red and $L^{[2]}$ highlighted in green). Therefore, a locality-aware schedule should fetch an equal number of elements from all bit-positions (Fig. 1b (right)).

Algorithm 2 shows our locality-aware scheduling algorithm where the data dependencies have been taken into consideration. The two innermost loops compute the multiplication across

Algorithm 2 Locality optimized bit-serial matrix multiplication.

1: **Input:** $m \times k$, w-bit matrix L and $k \times n$, a-bit matrix R
2: **Output:** $P = L \cdot R$
3: *# Bit-serial matrix multiplication*
4: **for** $r \leftarrow 1 \ldots m$ **do**
5: **for** $c \leftarrow 1 \ldots n$ **do**
6: **for** $d \leftarrow 1 \ldots k$ **do**
7: *# For all precisions of one element*
8: **for** $i \leftarrow 0 \ldots w-1$ **do**
9: **for** $j \leftarrow 0 \ldots a-1$ **do**
10: weight $= 2^{i+j}$
11: $P_{rdc} = P_{rdc} +$ weight $\times (L_{rd}^{[i]} \times R_{dc}^{[j]})$
12: *# Summation along the k dimension*
13: $P_{rc} = P_{rc} + P_{rdc}$

all precisions for each resulting element of the final matrix (P). One minor drawback of this schedule is that the weight (line 10) is constantly recomputed, potentially increasing the computational load.

Fig. 1c shows the access pattern of our locality-aware scheduling algorithm. Given the same resource constraints as described earlier (i.e., same on-chip memory size), two elements of each bit-matrix can be read and two partial results (P_{111} and P_{121}) can be completely computed before the next set of elements are read and computed. As seen in the figure, each element is only read a single time in contrast to the original schedule that causes the elements of the right-hand side matrices ($R_{dc}^{[j]}$) to be read multiple times.

III. The BISMO Accelerator

An accelerator optimized for performing binary matrix multiplication is not sufficient to take advantage of the locality-aware scheduling algorithm described in the previous section (Algorithm 2). BISMO [6] was designed for efficient computation of binary matrix multiplications but due to its software programmability it provides the necessary flexibility to evaluate a large variety of different scheduling algorithms.

The key execution unit of BISMO is the dot-product unit (DPU) as shown in Fig. 2. The DPU computes a partial binary dot-product through bit-wise logic AND operations and summation of these binary results using a popcount operation. D_k denotes the width of the vector performing a vectorized bit-serial execution, while A denotes the bit-width of the accumulator of the DPU.

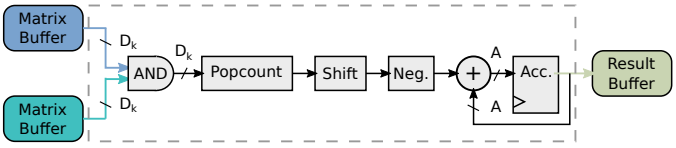


Fig. 2: The BISMO dot product unit (DPU).

Multiple DPUs are instantiated in an array-like structure referred to as a dot-product array (DPA). Each row and column of the DPA is fed by a matrix buffer, as shown in Fig. 3. The DPA is scaled by configuring the number of rows (D_m) and columns (D_n), which respectively alter the number of matrix buffers and DPUs. The DPUs are connected to the matrix

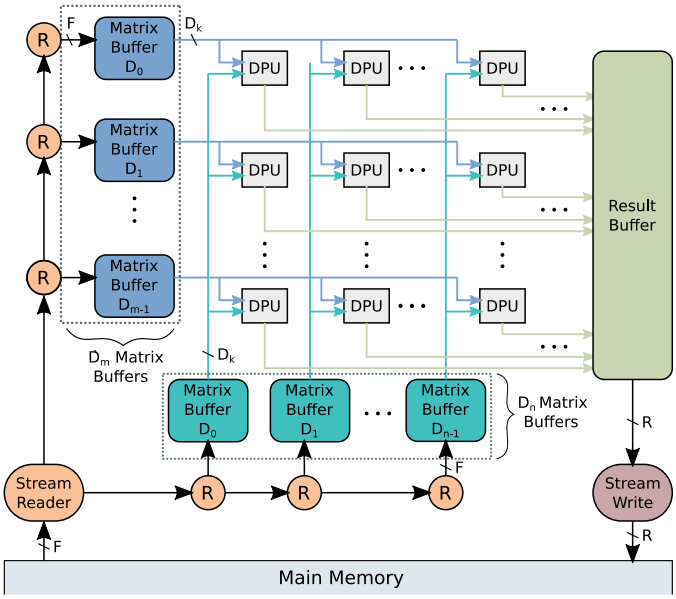


Fig. 3: The BISMO datapath.

buffers in a broadcast fashion. Note that the accumulator cannot load any results from main-memory and, therefore, the DPA must complete an inner product of a sub-matrix (a tile matching the DPA dimensions D_m, D_n) along the matrix dimension k before calculating the inner product of another sub-matrix. BISMO consists of a software controlled three stage pipeline design for (1) fetching binary matrices into the matrix buffers, (2) executing binary matrix multiplications on the DPA, and (3) writing the result back to memory.

For detailed information on BISMO and its software programmability please refer to the article by Y. Umuroglu et al. [6].

IV. Locality-Aware Bit-Serial Matrix Multiplication on BISMO

Algorithm 3 shows a detailed implementation for BISMO of the generic locality-aware algorithm described in Algorithm 2. Fig. 4 highlights the relationship between the complete computational space, the sub-block fetched to on-chip memory, and the sub-block that is consumed by the DPA. The algorithm has to consider the relationship between the size of the matrices (m, n, and k), the DPA dimensions (D_m, D_n, and D_k), the precision of the input matrices (w and a), and the size of the on-chip buffers (B_m and B_n). For ease of explanation it is assumed that the dimensions m, n, and k are evenly divisible by D_m, D_n, and D_k, respectively.

The algorithm assumes that the input matrices are larger than what can fit in the on-chip matrix buffers (D_i in Fig. 3) and that blocking (tiling) is necessary (line `9` and `10` in Algorithm 3). Each buffer is assumed to hold $m/B_m/D_m \cdot k \cdot w$ bits for the L buffer and $n/B_n/D_n \cdot k \cdot a$ bits for the R buffer, as illustrated by the on-chip memory block in Fig. 4. Since an inner product has to be completed due to the constraints given by BISMO, the on-chip memory block is formed giving priority to the k dimension (line `16`), before traversing across the m and

Algorithm 3 Vectorized bit-serial matrix multiplication on BISMO

1: **Input:** $m \times k$, w-bit matrix L and $k \times n$, a-bit matrix R
2: **Variable:** B_m number of blocks in the m dimension
3: **Variable:** B_n number of blocks in the n dimension
4: **Constant:** D_m number of DPU rows in the DPA
5: **Constant:** D_n number of DPU columns in the DPA
6: **Constant:** D_k vectorization width of DPU
7: **Output:** $P = L \cdot R$
8: *# Blocking of the computational space*
9: **for** $b_m \leftarrow 0 \ldots B_m - 1$ **do**
10: **for** $b_n \leftarrow 0 \ldots B_n - 1$ **do**
11: *# Fetch data to on-chip matrix buffers*
12: *# Iterate over the m and n dimension of a block*
13: **for** $r_i \leftarrow 0 \ldots (m/B_m)/D_m - 1$ **do**
14: **for** $c_i \leftarrow 0 \ldots (n/B_n)/D_n - 1$ **do**
15: *# Compute along the k dimension*
16: **for** $d_i \leftarrow 0 \ldots k/D_k - 1$ **do**
17: *# For all precisions*
18: **for** $i \leftarrow 0 \ldots w - 1$ **do**
19: **for** $j \leftarrow 0 \ldots a - 1$ **do**
20: weight $= 2^{i+j}$
21: *# Vectorized DPA computation*
22: *# Computed in a single cycle*
23: **for** $x \leftarrow 1 \ldots D_m$ **do**
24: **for** $y \leftarrow 1 \ldots D_n$ **do**
25: **for** $z \leftarrow 1 \ldots D_k$ **do**
26: $r = b_m \cdot B_m + r_i \cdot D_m + x$
27: $c = b_n \cdot B_n + c_i \cdot D_n + y$
28: $d = d_i \cdot D_k + z$
29: $P_{rc} = P_{rc} + \text{weight} \times (L^{[i]}_{rd} \times R^{[j]}_{dc})$

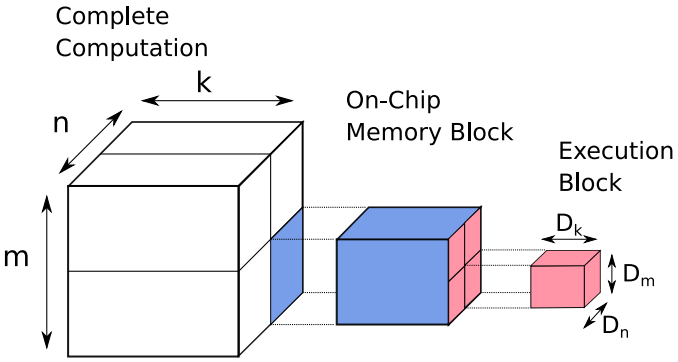


Fig. 4: The full matrix in relation to fetch and execute blocks

n dimension (line `13` and `14`). The bits from all precisions are read to assure locality (line `18` and `19`). The vectorized execution on a DPA occurs in a single cycle across an execution block (line `23-29`) as illustrated in Fig. 4.

The presented algorithm assumes that the matrix buffers can fit all the bits of one (or more, $D_m > 1$ or $D_n > 1$) row(s) or column(s). For input matrices with a k dimension larger than what would fit in the matrix buffers blocking of the k dimension is needed. For k-blocking D_m and D_n is set to one (line `9` and `10` becomes void) and the for-loop iterating over k is split into two loops (line `16`).

V. Evaluation

To highlight the improvements achieved by the locality-aware scheduling algorithm we present the required number of fetches based on the input-matrix dimensions relative to the

TABLE I: Required Fetches with respect to K_r

	Regular Bit-Serial				Locality Optimized				Improvement			
	Matrix precision ($w \times a$)											
K_r	1×1	2×2	3×3	4×4	1×1	2×2	3×3	4×4	1×1	2×2	3×3	4×4
1	2	6	12	20	2	4	6	8	0%	33%	50%	60%
2	4	16	36	64	4	8	12	16	0%	50%	67%	75%
3	6	24	54	96	6	12	18	24	0%	50%	67%	75%
4	8	32	72	128	8	16	24	31	0%	50%	67%	76%

TABLE II: Required fetches with respect to $B_m \times B_n$

	Regular Bit-Serial				Locality Optimized				Improvement			
	Matrix precision ($w \times a$)											
$B_m \times B_n$	1×1	2×2	3×3	4×4	1×1	2×2	3×3	4×4	1×1	2×2	3×3	4×4
1x1	2	6	12	20	2	4	6	8	0%	33%	50%	60%
2x2	6	20	42	72	6	16	24	32	0%	20%	43%	56%
3x3	12	42	90	156	12	36	54	72	0%	14%	40%	54%
4x4	20	72	156	272	20	64	96	128	0%	11%	38%	53%

available on-chip memory. The data has been obtained using the BISMO hardware-software co-simulator [7].

Table I shows the required fetches for a $w \cdot a$-bit matrix multiplication with respect to K_r. Here, K_r represents the size of k relative to the available size of a matrix buffer, i.e., $K_r == 1$ means that the bits of all the elements in the common dimension k for a single precision fit in on-chip memory (on-chip memory size $= k$ bits) and $K_r == 2$ that only half the bits fit. It can be seen that for binary matrix multiplications (1×1) the locality-aware algorithm does not provide any additional benefit. This is expected as all data fit in the on-chip buffers. The locality-aware algorithm starts to shine for larger bit-precision for which it reduces the number of fetches significantly. As expected, the benefit increases with the increased precision of the input matrices.

Table II shows the required number of fetches for a $w \cdot a$-bit matrix multiplication for a matrix of dimensions $B_m \times B_n$ relative to the DPA where $B_m = m/D_m$ and $B_n = n/D_n$, while $K_r = 1$. As expected, a similar behavior as for Table I can be seen when the matrix dimensions increases highlighting the importance of optimizing for multi-bit matrices that are significantly larger than the available on-chip memory.

VI. Conclusion

In this paper we propose a scheduling algorithm that improves the locality of vectorized bit-serial matrix multiplications. The algorithm eliminates redundant fetches for bit-serial matrix multiplication and has been implemented on the BISMO accelerator. The evaluation performed on BISMO shows that the locality-aware algorithm significantly reduces the number of fetches for multi-bit precision matrix multiplications.

References

[1] K. Asanović, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from Berkeley," Tech. Rep. UCB/EECS-2006-183, Dec 2006.

[2] M. H. S. Han and W. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proceedings of the International Conference on Learning Representations*, 2015.

[3] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv and Y. Bengio., "Quantized neural networks: Training neural networks with low precision weights and activations," *arXiv preprint arXiv:1609.07061*, 2016.

[4] P. Judd, J. Albericio, T. H. Hetherington, T. M. Aamodt, N. D. E. Jerger, R. Urtasun, and A. Moshovos, "Reduced-precision strategies for bounded memory in deep neural nets," *CoRR*, vol. abs/1511.05236, 2015.

[5] M. Zargham, *Computer architecture: single and parallel systems*. Prentice-Hall, Inc., 1996.

[6] Y. Umuroglu, L. Rasnayake, and M. Själander, "BISMO: A scalable bit-serial matrix multiplication overlay for reconfigurable computing," in *Proceedings of the Conference on Field Programmable Logic and Applications*, Aug. 2018.

[7] Y. Umuroglu, L. Rasnayake, and M. Själander, "Open-source implementation of BISMO." https://github.com/EECS-NTNU/bismo, 2018.