IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 16, 2022, accepted February 7, 2022, date of publication February 15, 2022, date of current version February 24, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3151876

Optimization of Multi-Core Accelerator Performance Based on Accurate Performance Estimation

SUNWOO KIM[1], YOUNGHO SEO[1], SUNGKYUNG PARK[2], (Senior Member, IEEE), AND CHESTER SUNGCHUNG PARK[1], (Senior Member, IEEE)

[1]Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, South Korea
[2]Department of Electronics Engineering, Pusan National University, Pusan 46241, South Korea

Corresponding author: Chester Sungchung Park (chester@konkuk.ac.kr)

This work was supported by the Samsung Research Funding and Incubation Center of Samsung Electronics under Project SRFC-IT1802-10.

ABSTRACT Multicore accelerators have emerged to efficiently execute recent applications with complex computational dimensions. Compared to a single-core accelerator, a multicore accelerator handles a larger amount of communication and computation simultaneously. Since the conventional performance estimation algorithm tailored to single-core accelerators cannot estimate the performance of multicore accelerators accurately, we propose a novel performance estimation algorithm for a multicore accelerator. The proposed algorithm predicts a dynamic communication bandwidth of each direct memory access controller (DMAC) based on the runtime state of DMACs, making it possible to estimate the communication amounts handled by DMACs accurately by taking into account the temporal intervals. The proposed algorithm is evaluated for convolutional neural networks and wireless communications. The experimental results using a pre-register transfer level (RTL) simulator shows that the proposed algorithm can estimate the performance of a multicore accelerator with the estimation error of up to 2.8%, regardless of the system communication bandwidth. These results were also verified by the hardware implementations on Xilinx ZYNQ. In addition, the proposed algorithm is used to explore a design space of accelerator core dimensions, and the resulting optimal core dimension provides performance gains of 10.8% and 31.2%, compared to the conventional multicore accelerator and single-core accelerator, respectively. The source code is available on the GitHub repository: https://github.com/SDL-KU/OptAccTile.

INDEX TERMS Communication bandwidth, direct memory access, multicore accelerator, performance estimation.

I. INTRODUCTION

With the rapidly increasing communication amount, general-purpose processors cannot achieve the performance and energy efficiency required for the latest applications. To address this problem, researchers have recently proposed custom hardware accelerators for specific applications.

Although the conventional hardware accelerators have significantly improved the performance and energy efficiency, there are limitations to the performance gains–particularly because that the hardware accelerator design is parameterized by the input/output dimensions. The performance and energy efficiency depend on the accelerator core dimensions and the compatibility with the input/output dimensions of the target application. In modern applications with high computational complexity, hardware accelerators optimized with one computational grid dimension cannot achieve the maximum performance and energy efficiency, owing to the dynamic underutilization of the hardware resources. For example, in convolutional neural network (CNN), a single-core accelerator [1] with an optimized design option has <24% dynamic hardware utilization. To resolve this problem, in [2]–[4], a multicore accelerator composed of several accelerator cores with different input/output dimensions was proposed. Each accelerator core is dedicated to input/output dimensions of each convolutional layer, which makes it possible to perform computational tasks with different computational dimensions in parallel, with significantly less idle hardware.

The associate editor coordinating the review of this manuscript and approving it for publication was Mitra Mirhassani.

To maximize the performance of the multicore accelerator, it is necessary to optimize the input/output dimensions (i.e., core dimensions) of the accelerator cores to match the workload. Shen *et al.* [2] maximize the performance of multicore accelerators by setting the core dimensions so that the execution time of each accelerator core is as close to that of other cores as possible. The performance of the optimized multicore accelerator is 3.8 times larger than that of a single-core accelerator [1] under the same hardware resource constraints. However, because it is assumed that all accelerator cores are computation limited, this conventional approach is effective only when the system communication bandwidth is sufficiently large. The performance gain, as expected in this conventional study, cannot be achieved in an actual implementation environment where the system communication bandwidth is limited by the on-chip bus or the dynamic random-access memory (DRAM).

Herein, we propose a performance estimation algorithm that can accurately estimate the performance of a multicore accelerator in all cases, considering the limited system communication bandwidth. The proposed algorithm assigns the communication bandwidth to multiple accelerator cores. In addition, the core dimensions of CNN accelerators are optimized using the proposed algorithm and verified the performance gain through pre-register transfer level (RTL) simulation and field-programmable gate array (FPGA) implementation.

The main contributions of this study are summarized as follows:

- We propose a performance estimation algorithm for a multicore accelerator that dynamically assigns a communication bandwidth to each direct memory access controller (DMAC) in accelerator cores. In contrast to the conventional estimation algorithm, the proposed algorithm predicts the communication times of individual DMACs and thus predicts whether the processing pass of each accelerator is limited by communication or computation. Therefore, the proposed algorithm accurately estimates the accelerator performance by considering a communication-limited case.
- We propose core dimensions that achieve the optimal performance when the system communication bandwidth is limited. The design space of accelerators is explored through the core dimension optimization of the multicore CNN accelerator using the proposed algorithm.

The remainder of this paper is organized as follows. Section II presents related works. Section III describes the multicore accelerator-based system considered in this paper and the core dimension for designing the accelerator. Moreover, in this section, we describe the technique for assigning the system communication bandwidth. In addition, Section IV presents backgrounds for performance estimation, including the hardware architecture of CNN accelerators. Section V introduces the proposed performance estimation algorithm and Section VI describes the core dimension optimization using the proposed algorithm. Section VII presents the optimization process of the multicore CNN accelerator for AlexNet. In this case, Xilinx Zynq implementation is assumed and pre-RTL simulation results for the corresponding core dimensions are presented. Section VIII presents the results of the FPGA implementation using the optimal core dimensions mentioned above. Finally, Section IX concludes the paper and discusses the future work.

II. RELATED WORKS

To achieve high performance, it is better to maximize the utilization of hardware resources and minimize the memory access between the accelerators and off-chip DRAM. Both hardware resources and memory access of accelerators are affected by accelerator core dimensions. However, in the hardware implementation stage, it is difficult to estimate both the computation time and the communication time by considering the system communication bandwidth which is affected by selection of accelerator core dimensions. To solve this problem, many studies propose performance estimation algorithms for pre-implementation stages.

Many studies [5], [6], [24] that estimates the communication time of a multicore accelerator generally estimated the communication time of a hardware architecture in which multiple masters access a slave (i.e., off-chip memory) through an on-chip bus. Kim *et al.* [5] propose a static estimation algorithm that utilizes the memory trace information and the execution schedule based on a queuing model. This algorithm estimates the bus contention and provides an accurate communication time estimation. However, most hardware accelerators execute computation and communication in parallel, and the execution time of the two operations cannot always be the same, and there are situations in which one operation waits for the completion of the other operation. The static estimation algorithm [5] cannot reflect the communication delay interruption owing to the computation of the hardware accelerator. Choi *et al.* [6] propose a performance estimation of a hardware architecture in which multiple processing elements (PEs) share off-chip memory according to the on-chip bus of the AXI protocol. The parallel operation time of the communication and computation is estimated through the minimum bus occupation cycles and the maximum value of the individual latencies of each PE. However, they focus on determining the critical path delay within a short simulation time, and thus the performance estimation accuracy decreases as the communication accessing slaves (i.e., off-chip memory) increases. Sinha *et al.* [24] introduce a design exploration framework for optimizing the memory subsystem of a multicore accelerator system. The performance error rate obtained using their framework is up to 4.12%, which is not significantly different from the performance estimation algorithm proposed in this paper. However, this framework [24] focuses on optimizing the memory subsystem, i.e., the memory configuration and data allocation. In addition, they assume that the sizes of the on-chip buffer

inside accelerator cores are uniformly distributed. Thus, it is limited to multicore accelerator configurations with various core dimensions by dividing them differently for each core. In other words, Sinha *et al.*'s framework differs from the algorithm proposed in this paper in that the latter optimizes the accelerator core using performance estimation.

The computation time estimation of hardware accelerators also have been studied extensively for different target applications. For CNN accelerators, many studies [1], [2], [7], [8] introduce performance estimation algorithms based on static analysis. Zhang *et al.* [1] exploit the computation to communication (CTC) ratio to determine the optimal design point. The computation roof was measured using the total number of operations performed by the accelerator and the number of operations. The optimal design space suitable for the given hardware resource constraints was selected with a graph consisting of the measured CTC ratio, computation roof, and communication bandwidth roof. Moreover, Shen *et al.* [2] introduce a performance estimation algorithm which follows most of ideas in the algorithm of Zhang *et al.* [1], which is tailored to single-core accelerators. For this reason, it is difficult to consider the data communication delay between the accelerator cores in the aforementioned two studies. Therefore, when the system communication bandwidth is limited, it is difficult to expect that the estimated performance matches well with the performance of the actual implementation. Ma *et al.* [7] focus on the estimation of the communication time using the models of DRAM access, latency, and on-chip buffer access. Modeling is formulated by the design strategies and design parameters (i.e., core dimensions) of loop tiling and unrolling, which results in the estimation error within 3%. Zhao *et al.* [8] introduce a deep neural network (DNN)-chip predictor that estimates the energy, throughput, and latency of a DNN accelerator. Moreover, the authors [8] proposed a performance estimation based on the number of MAC operations of the MAC units and on the number of access operations between various memory hierarchies (DRAM, global buffer, and register files), assuming a row-stationary dataflow-based CNN accelerator [15]. In particular, the performance estimation algorithm considers the communication bandwidth between each memory hierarchy in detail. However, the algorithm in [8] estimates the communication time by considering only the number of times the data are loaded or stored for each memory hierarchy assuming a single-core accelerator. In addition, this algorithm does not consider the corner cycles where the communication time and computation time are not fully pipelined. As a result, this algorithm is outperformed by the performance estimation algorithm using dynamic communication bandwidth and considering the amount of data transmission for each direct memory access (DMA) interval, in terms of the estimation accuracy.

III. MULTI-CORE ACCELERATORS

This section describes the overall system assumed in this study, especially the multicore accelerator.

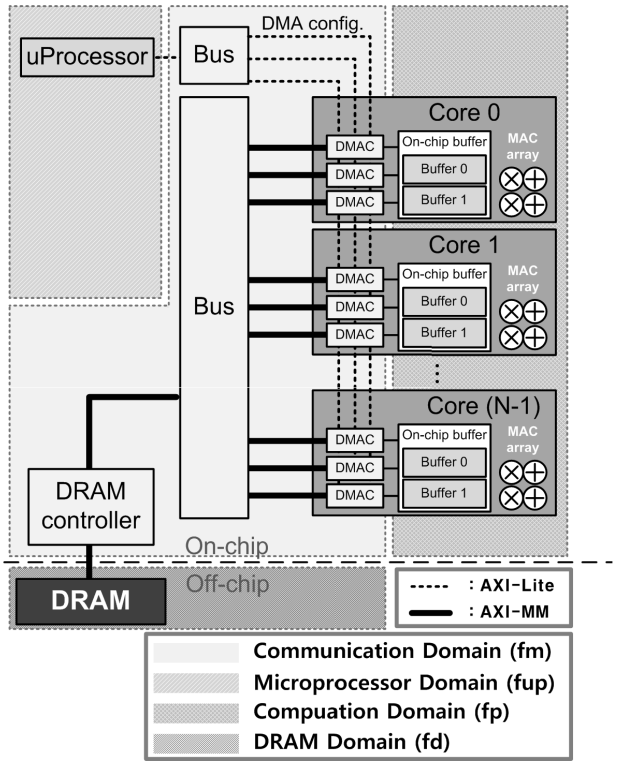


FIGURE 1. Overall system with a multi-core accelerator.

A. SYSTEM ARCHITECTURE UNDER CONSIDERATION

Fig. 1 shows the system architecture under consideration, consisting of a multicore accelerator, microprocessor, on-chip buses, and DRAM subsystem. The accelerator accesses DRAM subsystem through an on-chip bus. In detail, one or more DMACs inside each accelerator access DRAM subsystem as a bus master or as a bus slave. The microprocessor sets the source and destination addresses of each DMAC which is responsible for transferring data from/to each accelerator.

In this study, it was assumed that accelerators communicate with the off-chip DRAM using DMACs, as in the accelerators [3], [7]–[11] of conventional studies. Specifically, each accelerator core loads the accelerator input from DRAM to the on-chip buffer or stores the accelerator output from the on-chip buffer to DRAM through one or more dedicated DMACs. In this process, loop tiling is required because of the size limitation of the on-chip buffer [1]–[3], [7], [12], [13]. The arithmetic units of each accelerator core perform arithmetic operations using the accelerator inputs stored in the on-chip buffer and store the accelerator outputs in this same buffer. The on-chip buffer is implemented with double buffering [1], [2], [8], [10], [14] to overlap the communication and computation. One buffer communicates with DRAM, whereas the other buffer proceeds with the arithmetic operations and computation. This process is repeated in a ping-pong manner. The number of DMACs is implementation

dependent and three DMACs are assumed for an accelerator core in Fig. 1.

The system communication bandwidth is determined by the on-chip bus and the system parameters of DRAM subsystem including DRAM controller. For example, the burst length of the on-chip bus, the number of ports or bitwidth of the bus [1], [10], etc., are included. In addition, in general, the clock frequencies of the on-chip bus and DRAM subsystem may differ from those of the accelerator [15]–[19]. In this study, it is assumed that the clock frequency is divided into the computation clock domain (**fp**) and communication clock domain (**fm**) of the accelerator, the microprocessor clock domain (**fup**) and DRAM clock domain (**fd**), as shown in Fig. 1. When considering this, it is found that the given system communication bandwidth differs greatly depending on the system, and unlike the situation assumed in [2], the required system communication bandwidth may not be satisfied in the system. Moreover, when multiple different DMACs of multiple accelerator cores transmit bus requests simultaneously, the communication bandwidth assigned to each core tends to decrease with the number of active DMACs owing to bus contention. More importantly, the communication bandwidth assigned to each core dynamically varies depending on the set of active DMACs. For example, assuming bus arbitration with equal priority, it is found that the communication bandwidth assigned to each core of the active DMAC, is inversely proportional to the number of active DMACs [20].

B. ACCELERATOR CORE DIMENSION

The accelerator core is designed by assuming the hardware resources of a specific dimension for the accelerator input/output. The actual workload dimension, i.e., the accelerator input/output dimension, may be different. For example, in the case of matrix-vector multiplication, the number of rows and columns in the matrix may be different from those assumed by the accelerator core. To solve the mismatch between the accelerator input/output dimension and the core dimension, we mainly designed the accelerator core with loop tiling determined by the tile size according to the core dimension. When the accelerator input/output dimension is smaller than the tile size or is not an integer multiple of the tile size, the hardware resources of the accelerator core are severely underutilized. However, partitioning a given hardware resource into multiple accelerator cores of various dimensions can solve the severely underutilized hardware resources mentioned above.

Fig. 2 illustrates the performance gain of a multicore accelerator compared with that of a single-core accelerator in the case of matrix-vector multiplication. In the figure, it is assumed that the total number of multipliers and adders of the multicore accelerator is the same as that of the single-core accelerator. When computing three matrix-vector multiplications with different dimensions, the single-core accelerator operates with low hardware resources, as shown in Fig. 2(a). However, as shown in Fig. 2(b), using a multicore accelerator shows high hardware resource utilization, which

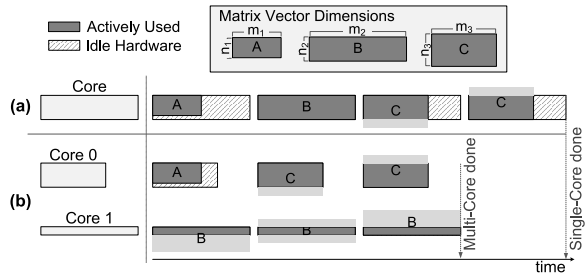


FIGURE 2. Operations of the (a) single-core accelerator and (b) multi-core accelerator on matrix-vector multiplications.

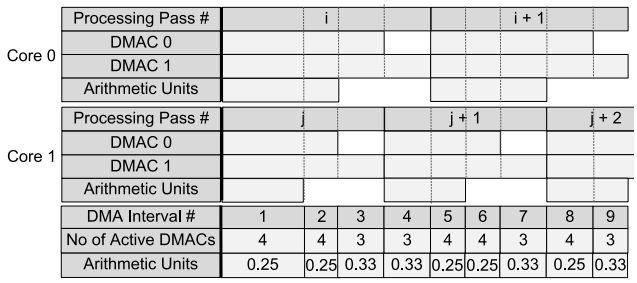


FIGURE 3. Modeling of dynamic communication bandwidth depending on execution time.

indicates a performance improvement. This means that, when input/output dimensions are matched using tiling, multiple accelerator cores designed with small dimensions are more appropriate than single accelerator cores designed with large dimensions.

C. ACCELERATOR COMMUNICATION BANDWIDTH

Owing to the size limitation of the on-chip buffer, loop tiling divides the task of the accelerator into multiple processing passes and processes them iteratively. Specifically, in one processing pass, DMACs load a small subset of the accelerator inputs from DRAM and store them in an on-chip buffer, and the arithmetic units perform arithmetic operations. Subsequently, DMACs store the corresponding accelerator output from the on-chip buffer to DRAM. As mentioned earlier, owing to the overlapping of the computation and communication through the double buffering, the duration of each processing pass is determined by the maximum values of the computation and communication times. In this paper, when the duration of the processing pass is determined by the computation time, the accelerator is defined as computation limited, and when it is determined by the communication time, the accelerator is defined as communication limited.

Fig. 3 illustrates the computation and communication for each processing pass in a multicore accelerator composed of two accelerator cores. It is assumed that each accelerator core is equipped with two DMACs (DMAC 0 and DMAC 1) and is communication limited in all processing passes. As shown in the figure, the number of active DMACs varies with time, and in particular, the processing pass of one accelerator core

```
for (m = 0; m < M; m++) {
  for (c = 0; c < C; c++) {
    for (e = 0; e < E; e++) {
      for (f = 0; f < F; f++) {
        for (r = 0; r < R; r++) {
          for (s = 0; s < S; s++) {
            OA[m][e][f] +=
              IA[c][e+r][f+s]*W[m][c][r][s];
}}}}}}
    post_processing(OA);
```

FIGURE 4. Pseudocode of the convolutional layers.

is generally not synchronized with them of other cores, and thus, it varies across processing passes. With the set of active DMACs, each processing pass is again divided into several sub-passes, which are defined as DMA intervals in this paper. Assuming bus arbitration with equal priority, it is found that the communication bandwidth assigned to each active DMAC in each DMA interval is inversely proportional to the number of active DMACs. The number of active DMACs of each DMA interval and the communication bandwidth per core (beats per cycle) are shown in the bottom of Fig. 3. For example, in DMA intervals 1 and 2, because all four DMACs are active, each DMAC is assigned 0.25 beat/cycle of the system communication bandwidth. On the other hand, in DMA interval 3, DMAC 0 of Core 1 is no longer active because there is no remaining communication amount. Therefore, three DMACs are active, and each DMAC is assigned 0.33 beat/cycle of the system communication bandwidth. Similarly, in DMA interval 4, DMAC 0 of Core 0 and DMAC 1 of Core 1 have no remaining communication amount. Core 1 operates to handle the new processing pass, and thus both DMACs become active. Therefore, DMAC 0 of Core 0 and all DMACs of Core 1 are active and the system communication bandwidth of 0.33 beat/cycle is assigned to each of these active DMACs. In summary, the communication bandwidth per core varies with DMA intervals and is inversely proportional to the number of active DMACs. More importantly, the communication time of each core generally varies across processing passes.

IV. CNN ACCELERATOR EXAMPLE

In this section, CNN accelerators are considered as an example of the multicore accelerator described above. Specifically, we describe the hardware architecture of the accelerator core and the effect of the core dimension on computation and communication.

A. ACCELERATOR CORE HARDWARE

Convolution is the main operation in a CNN that requires a number of multiply-and-accumulate (MAC) operations that require input activations (IAs) and filter weights (Ws) to produce output activations (OAs). Each convolutional layer consists of six nested loops, as shown in Fig. 4. For the input image, **C** input feature maps are convolutional with **M** filters to generate **M** output feature maps. The height and width of each data type, namely, the input feature map, filter, and output feature map, are denoted by **H/W**, **R/S**, and **E/F**, respectively.

```
for (e = 0; e < E; e += TE) {
  for (f = 0; f < F; f += TF) {
    for (m = 0; m < M; m += TM) {
      for (c = 0; c < C; c += TC) {
        ibuf = IA[c:c+TC-1][e:e+TH-1][f:f+TW-1];
        wbuf = W[m+tm :m+TM-1][c:c+TC-1][:][:];
        for (r = 0; r < R; r++) {
          for (s = 0; s < S; s++) {
            for (te = 0; te < TE; te++) {
              for (tf = 0; tf < TF; tf++) {
                for (tm = 0; tm < TM; tm++) {
                  #pragma HLS UNROLL
                  for (tc = 0; tc < TC; tc++) {
                    #pragma HLS UNROLL
                    ix =ibuf [tc][te+r][tf+s];
                    wx =wbuf[tm][tc][r][s];
                    obuf[tm][te][tf] += ix*wx;
                  } }
        } } } }
      }
      OA[m:m+TM-1][e:e+TE-1][f:f+TF-1] = obuf;
} } }
post_processing(OA);
```

FIGURE 5. Pseudocode for NLR with loop tiling.

We assume a no-local-reuse (NLR) dataflow-based CNN accelerator synthesized using Xilinx Vivado HLS, as illustrated by the pseudocode in Fig. 5. As mentioned earlier, we assume a loop tiling-based CNN accelerator using the HLS UNROLL pragma, and thus data communication and computation are executed in parallel. Loop tiling is parameterized by the size of the tiles (**TE**, **TF**, **TM**, and **TC**). The on-chip buffer assumes double buffering to overlap the operation time and data transmission time. Note that **ibuf**, **wbuf**, and **obuf** represent the input activations, filter weights, and output activations stored in the on-chip buffer, respectively. The accelerator accesses off-chip memory for IAs, Ws, and OAs using IA DMAC, W DMAC, and OA DMAC, respectively. Similarly, the input activations, filters weights and output activations stored in local registers are denoted by **ix**, **wx**, and **ox**, respectively. When the input activations and filter weights are transferred to the on-chip buffer, the accelerator consisting of multiple MACs transmits the **TC** input activations and **TM** partial sums obtained through the **TC**·**TM** filter weights to the on-chip buffer. After the convolutional operation for the tile is completed, the output activation is saved in the off-chip DRAM. For the partial sum operation, activations and weights must be stored in on-chip buffers, and thus the size of the on-chip buffer is determined by the size of the tile. When the convolutional operation on the tile is completed, the output activations are stored in the off-chip memory. The remaining convolutional loops that are not unrolled are performed by repeating the aforementioned operation, as illustrated in Fig. 5.

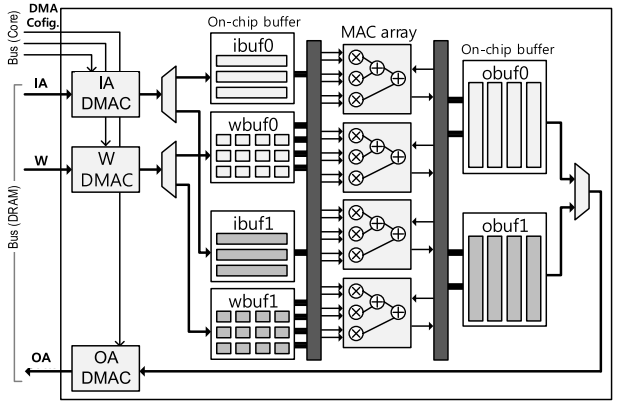


FIGURE 6. CNN accelerator for NLR with loop tiling.

Fig. 6 is a block diagram of the CNN accelerator (TC = 3 and TM = 4) synthesized by the HLS code. The MAC array consists of TC·TM MAC units and performs the same number of MAC operations in each cycle. In the figure, we assumed three DMACs, namely, IA-DMAC for input activations, W-DMAC for filter weights and OA-DMAC for output activations. Each DMAC is responsible for transferring data from/to the off-chip DRAM via an on-chip bus. DMACs communicate data through connections with on-chip buffers implemented as static RAMs (SRAMs). Since each on-chip buffer consists of double buffers, a pair of on-chip buffers are utilized exclusively for each data type.

B. IMPACT OF LOOP TILING

As mentioned above, one convolutional layer is iteratively processed as multiple processing passes owing to loop tiling. Assuming the pseudocode in Fig. 5, we can obtain the total number of processing passes as

$$Np = \lceil \frac{M}{TM} \rceil \times \lceil \frac{C}{TC} \rceil \times \lceil \frac{E}{TE} \rceil \times \lceil \frac{F}{TF} \rceil \quad (1)$$

Equation (1) shows that the number of processing passes increases as tile sizes decrease. In addition, as mentioned above, double buffering enables overlap of computation, and thus the duration of each processing pass is determined by the maximum values of the computation and communication times. Here, the communication time depends on the communication amount as well as on the communication bandwidth per core. The required communication amount in each processing pass is given as

$$Ni = TC \times TH \times TW \quad (2)$$
$$Nw = TM \times TC \times R \times S \quad (3)$$
$$No = TM \times TC \times TF \quad (4)$$

where **Ni**, **Nw** and **No** represent the communication amount for the input activations, filter weights and output activations, respectively.

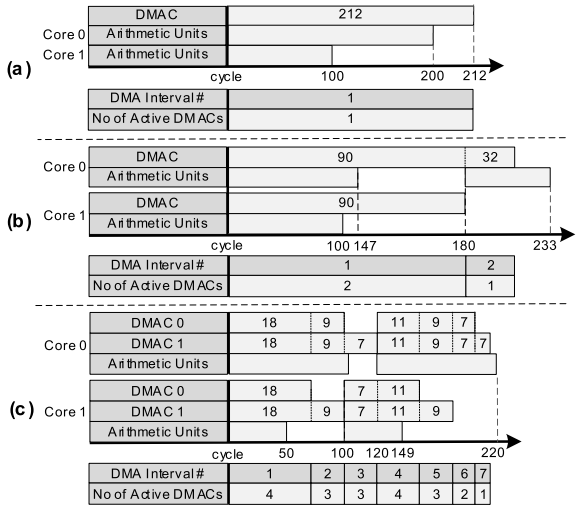


FIGURE 7. Examples of the performance estimation algorithms: (a) conventional estimation, (b) per-core estimation and (c) (proposed) per-DMAC estimation.

On the other hand, the computation time is directly determined by the tile size as

$$Tp = TE \times TF \times R \times S \quad (5)$$

V. PERFORMANCE ESTIMATION ALGORITHM

In this section, we explain how the proposed performance estimation algorithm estimates the performance of a multi-core accelerator by estimating the time-varying communication bandwidth per core.

A. CONVENTIONAL PERFORMANCE ESTIMATION

Shen *et al.* [2] minimize the critical delay among computation times of accelerator cores to maximize the performance of the multicore accelerator. In addition, they mainly consider that all the accelerator cores are computation limited. The required system communication bandwidth for the optimum core dimension is calculated using the total communication amount of all cores. However, in a communication-limited case, it is difficult to estimate the communication time accurately using such a conventional method, because the remaining communication amount of each DMAC cannot be tracked.

Fig. 7 illustrates the estimations of communication and computation times for two cores each of which includes two DMACs. It is assumed that DMAC 0 and DMAC 1 of Core 0 transfer communication amounts of 54 and 68 pixels/weights, respectively, and DMAC 0 and DMAC 1 of Core 1 transfer 36 and 54 pixels/weights, respectively. Additionally, computation times of 100 and 200 cycles are assigned to Core 0 and Core 1, respectively. As shown in Fig. 7(a), in the conventional performance estimation method, it is assumed that all accelerator cores communicate through one DMAC. Thus, it is clearly shown that the conventional performance

Algorithm 1 Performance Estimation

```
    input: BW, Nc, Np[Nc], N1[Nc], N2[Nc], Tp[Nc]
    output: Ttotal
    Initialization: N1r = 0, N2r = 0, Tpr = 0, Cp = 0;
 1: while(1)
 2:   for i = 1: Nc
 3:     if(Cp[i] <= Np[i])
 4:       if(N1r[i] == 0 && N2r[i] == 0 && Tpr[i] == 0)
 5:         N1r[i] = N1[i];
 6:         N2r[i] = N2[i];
 7:         Tpr[i] = Tp[i];
 8:         Cp[i] + = 1;
 9:       end
10:     end
11:   end
12:   Nr = nonzeros([N1r N2r]);
13:   Nad = size(Nr);
14:   Tmh = min(Nr) * Nad / BW;
15:   Tph = min(nonzeros(Tpr));
16:   Ti = min(Tmh, Tph);
17:   if (Tmh = { })
18:     Nd = 0;
19:   else
20:     Nd = Ti / Nad * BW;
21:   end
22:   Nir- = Nd;
23:   Nwr- = Nd;
24:   Tpr = max(Tpr – Ti, 0);
25:   Ttotal + = Ti;
26:   if (Cp == Np)
27:     break;
28:   end
29: end
```

estimation fails to accurately estimate the actual performance (given in Fig. 7(c)).

B. PROPOSED PERFORMANCE ESTIMATION

By keeping track of the communication amount of each accelerator core, it is possible to estimate the performance of the multicore accelerators more accurately. Specifically, as shown in Fig. 7(b), the communication time of each core is estimated by considering the communication bandwidth of each core according to the number of active DMACs in the DMA interval. For example, in DMA interval 1, each core is assigned 0.5 beat/cycle of the system communication bandwidth. However, the per-core performance estimation method still cannot accurately estimate the actual performance, since it is assumed that each accelerator core communicates through one DMAC. Fig. 7(b) shows the performance when it is assumed that the computation of each accelerator core is performed during the communication time.

In this paper, we propose a per-DMAC performance estimation method that tracks the computation and communication of each DMAC. As shown in Fig. 7(c), the per-DMAC performance estimation method can accurately estimate the communication amount by considering the communication bandwidth of each accelerator core in each DMA interval. Moreover, in contrast to the conventional performance estimation algorithms, it is possible to track the progress of the processing pass for the proposed algorithm, on the basis of the remaining communication amount of each DMAC. For example, in Fig. 7(c), the remaining communication amount of DMAC 1 for Core 1 becomes zero in DMA interval 2, and a new processing pass starts with DMA interval 3.

Algorithm 1 describes the procedure for estimating the performance of a multicore accelerator using the proposed performance estimation algorithm for a given core dimension. For simplicity, it is assumed that each core is assigned only one task (one layer in the case of a CNN). Moreover, it is assumed that one accelerator core is equipped with only two DMACs and that all processing passes have the same communication amount and computation time. The proposed performance estimation algorithm takes the system communication bandwidth (**BW**), the number of accelerator cores (**Nc**), the number of processing passes (**Np**), the communication amount per processing pass (**N1** and **N2**), and the computation time per processing pass (**Tp**) as inputs, and then it estimates and outputs the total execution time (**Ttotal**). In the algorithm, it is assumed that **Np**, **Ni**, **Nw**, and **Tp** can be different for each core, and they are calculated using Equations (1)–(5) in the case of the CNN accelerator. The algorithm first updates the remaining communication amount (**N1r** and **N2r**) and computation time (**Tpr**) that must be processed in the current processing pass for each accelerator core in the current DMA interval, and then it updates the pass number (**Cp**) of the current processing pass (lines 2–11). These are updated each time a new processing pass is started, until all the processing passes of the corresponding core are complete (line 3); i.e., they are repeated every time the communication and computation of the processing pass are completed (line 4). Subsequently, the number of active DMACs (**Nad**) of the current DMA interval is calculated on the basis of the remaining communication amount (**Nr**) (lines 12–13). Moreover, the required time for the DMAC with the smallest remaining communication amount to complete the communication is calculated as the hypothetical communication time (**Tmh**) (line 14), where all active DMACs have the same communication bandwidth per core; that is, it is assumed that each DMAC is assigned **BW/Nad** of the system communication bandwidth. In addition, the required time for the core with the least remaining computation time to complete the computation is calculated as the hypothetical computation time (**Tph**) (line 15). Moreover, as indicated by Equation (7), the minimum of **Tmh** and **Tph** is calculated as the actual duration (**Ti**) of the current DMA interval (line 16). Then, as indicated by line 14, it is assumed that all active DMACs are assigned the same communication bandwidth,

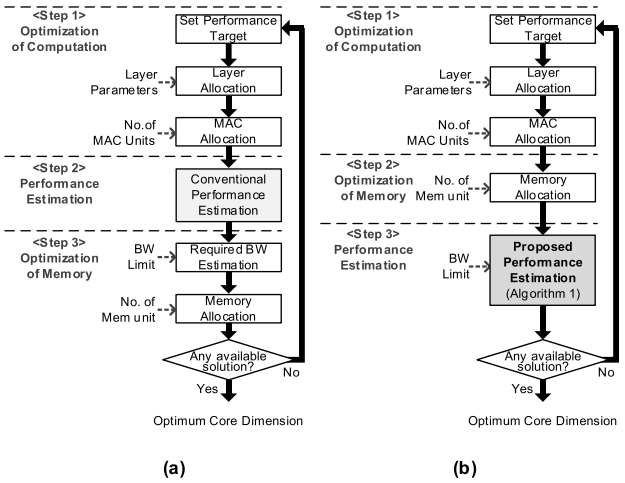


FIGURE 8. Optimization of the accelerator core dimensions: (a) conventional optimization and (b) proposed optimization.

and the actual communication amount (**Nd**) of the current DMA interval is calculated (lines 17–21). Finally, the remaining communication amount (**N1r** and **N2r**), computation time (**Tpr**), and total execution time (**Ttotal**) that were calculated are updated (lines 22–25), and the next DMA interval in the same process is calculated. This process is repeated until all the processing passes for all the cores have been completed (line 26).

In summary, the proposed performance estimation algorithm tracks the computation and communication of each accelerator core in each DMA interval to estimate the performance of a multicore accelerator. Using this algorithm, we can calculate whether a specific processing pass of each core is computation limited or communication limited. By calculating the assignment of the communication bandwidth per core using the number of active DMACs, we can accurately estimate the communication amount of each DMA interval.

VI. OPTIMIZATIONS OF THE ACCELERATOR CORE DIMENSIONS

In this section, we describe the optimization of the core dimension for a multicore CNN accelerator using the proposed performance estimation algorithm. We focus on the optimal network partitioning and tile sizes that satisfy the specified hardware resource constraints for optimization.

Fig. 8 shows the process of core dimension optimization, whereby the optimal core dimensions are determined using the parameters and hardware constraints of CNN convolutional layers. As mentioned earlier, the core dimensions include the number of accelerator cores and design parameters, such as the layer allocation on cores and tile sizes. The hardware constraints include the number of MAC units, number of memory units, and bandwidth limit. When hardware accelerators are implemented in the FPGA environment, MAC units and memory units are mapped to digital signal processing (DSP) slices and block RAMs (BRAMs), respectively. The numbers of DSP slices and BRAMs are determined by selecting the FPGA device for implementation. The bandwidth limit is determined by the ratio of clock domains, e.g., the accelerator computation clock domain (**fp**) and the accelerator communication clock domain (**fm**).

Fig. 8(a) illustrates the conventional optimization process for the core dimension. The process is divided into three steps. In step 1, the computation of the core dimensions is determined. First, the performance target is determined (set performance target), and convolutional layers allocated in accelerator cores are determined (layer allocation). Subsequently, MAC units are assigned to each accelerator core, i.e., the tile sizes of TM/TC are determined (MAC allocation). In step 2, considering the number of MAC units of each accelerator core, performance estimation is performed by calculating the total computation time required for the convolutional layers. Finally, in step 3, the memory sizes of the accelerator cores are determined. The peak communication bandwidth required for the target performance is obtained to ensure that the computation is limited by preventing the required bandwidth from exceeding the bandwidth limit (required BW estimation). Additionally, the memory size inside each accelerator core is determined by receiving the maximum number of memory units, where the tile sizes of TE/TF are determined (memory allocation). However, if the estimated performance of the core dimension obtained through the process does not satisfy the performance target or if the required bandwidth exceeds the bandwidth limit, the algorithm returns to Step 1 with the reduced performance target to explore the core dimensions. If core dimensions that satisfy both the performance target and the bandwidth limit are found, the optimization process ends.

Fig. 8(b) shows the proposed optimization process for the core dimensions. It is divided into three steps, similar to the conventional process, but the order is different. Step 1 is identical to that in the conventional optimization process; however, in Step 2, memory is optimized instead of performance estimation. Performance estimation takes place in Step 3, after the tile sizes corresponding to the allocation of MAC units and memory units have been determined. However, because the performance is not always assumed to be composition-limited in the proposed performance estimation algorithm, the performance estimation step is the last step in the proposed process, receiving memory optimization results. Therefore, the communication time is independent of the computation time, and the communication bandwidth is assumed to satisfy the bandwidth limit. Instead of estimating the required bandwidth, performance degradation due to the bandwidth limit is considered later. In the last step, the accelerator performance is estimated, and when the performance target is not satisfied, the performance target is reduced to explore the optimal core dimensions.

The main difference between the two optimization processes is the assumption as to whether the performance is always computation limited. In the conventional optimization

process, it is assumed that the communication time is greater than or equal to the communication time for all accelerator cores. However, there may be optimal core dimensions with a communication time longer than the computation time. The proposed process considers the communication-limited case and utilizes the performance estimation algorithm to obtain the core dimensions with the best performance and the same hardware constraints as the conventional process.

VII. EXPERIMENTAL RESULTS

In this section, we evaluate the accuracy of the proposed performance estimation algorithm through pre-RTL simulation. Additionally, we evaluate the performance of a multicore accelerator that optimizes the core dimension using the proposed performance estimation algorithm. The proposed performance estimation algorithm is applicable to all multicore accelerators, and in this section, accelerators for CNN and wireless communications are assumed.

A. EXPERIMENTAL SETUP

AccTLMSim [21] is a pre-RTL accelerator simulator based on SystemC transaction-level modeling (TLM). In addition, the pre-RTL simulator is incorporated with DRAM simulators such as DRAMsim2 [22]. In this study, the accelerator model in the simulator is extended to the multicore accelerator as illustrated in Fig. 1. Moreover, the conventional DMAC is replaced by the Scatter-Gather DMAC, which generates the source and destination addresses of each processing pass using address offsets. When using the conventional DMAC of AccTLMSim [21], the microprocessor should set the source and destination addresses to each DMAC at the starting point of the processing pass of each accelerator core. However, in a multicore accelerator, it is difficult for one microprocessor to simultaneously control the DMACs of all accelerator cores. Therefore, before the first processing pass of each accelerator core starts, all DMACs receive not only the address but also the number of processing passes, number of communication amounts per processing pass, and tile sizes from the microprocessor and store them in the control register. The DMAC generates the source and destination addresses required for each processing pass based on the information stored in the control registers without receiving any command from the microprocessor. As mentioned in Section III, the system considered in this study was set as the computation clock domain (**fp**) of the accelerators excluding the DMACs, communication clock domain (**fm**) of the on-chip bus and DRAM controller, processor clock domain, and DRAM clock domain, as shown in Fig. 1.

Table 1 presents the target network, hardware resource constraint, clock domain, and AMBA AXI parameters assumed for the simulation. Note that the hardware resource constraints of the Xilinx Virtex-7 FPGA (690T) are assumed for performance comparison with the multicore accelerator proposed in [2]. The computation clock domain of the accelerator was assumed to be 100 MHz, as per the assumption of Shen *et al.* [2]. The communication clock domain

TABLE 1. Design space for evaluating the proposed performance estimation.

Target Network		AlexNet
Hardware Resource Constraint	No. MAC Units	<2,880
	No. Memory Units	<1,238
Clock Domain (GHz)	Microprocessor (fup)	0.6
	Computation (fp)	0.1
	Communication (fm)	[0.1 : 0.02: 0.4]
	DRAM Device (fd)	0.5
AMBA AXI	Burst Length	16 beats
	No. Outstanding Transactions	4

was swept from 100 to 400 MHz at 20-MHz intervals, and the performance of the multicore accelerator was measured according to the system communication bandwidth. The number of ports or bitwidth of the bus is assumed to be values where the system communication bandwidth is 1 pixel/cycle or 1 weight/cycle with the communication clock frequency of 100 MHz. Therefore, the system communication bandwidth reached a maximum of 4 pixels/cycle or 4 weights/cycle.

Table 2 presents the core dimensions considered in experiments for the CNN accelerators, all of which satisfied the hardware resource constraints listed in Table 1. It is assumed that the accelerator hardware is implemented on a Xilinx Zynq-7045 SoC chip (an FPGA device including ARM core processors), and the corresponding optimization process of the core dimensions was conducted. The Zynq-7045 contains 900 DSP slices and 545 BRAMs. In this study, it was assumed that the CNN accelerators are implemented using 70% of the entire hardware resources. Applying the hardware constraints to the optimization process shown in Fig. 8(b) makes it possible to obtain the core dimensions from CD1 to CD4, including the optimal core dimension. These core dimensions were verified not only by the proposed performance estimation algorithm but also by the pre-RTL simulator and Zynq platform implementation. As described previously, the differences between the core dimensions can be observed in the results of the proposed algorithm and the simulator. In addition to the conventional core dimensions for single-core [1] and multicore accelerators [2], the four core dimensions (CD1 to CD4) proposed in this paper are described. These are the core dimension optimizations discussed in Section VI. CD1 and CD3 consist of five accelerator cores, and CD2 and CD4 consist of six accelerator cores.

B. PERFORMANCE ESTIMATION

In this subsection, we discuss the accuracy of the performance estimation for all system communication bandwidths, targeting the core dimensions presented in Table 2. Specifically, the estimation results of the three performance estimation algorithms shown in Fig. 7, including the proposed performance estimation algorithm, are compared with the simulation results of the pre-RTL simulator.

Fig. 9 presents the performance estimation accuracy of the CNN accelerator experiments. In most cases, the proposed per-DMAC estimation method exhibits a significantly

TABLE 2. Optimize core dimensions of multi-core accelerator.

Core Dimensions	Process Core	[TM, TC]	[TE, TF]	Layers
CD 1	0	[48, 1]	[14, 19]	1a
	1	[48, 1]	[14, 14]	1b
	2	[128, 2]	[27, 27]	2a, 2b
			[13, 13]	5a, 5b
	3	[64, 2]	[13, 13]	3a, 3b
	4	[96, 1]	[13, 13]	4a, 4b
CD 2	0	[48, 1]	[14, 19]	1a
	1	[48, 1]	[14, 19]	1b
	2	[128, 1]	[27, 27]	2a, 2b
			[13, 13]	5a, 5b
	3	[64, 1]	[13, 13]	3a
	4	[64, 1]	[13, 13]	3b
	5	[96, 1]	[13, 13]	4a, 4b
CD 3	0	[48, 1]	[14, 19]	1a
	1	[48, 1]	[14, 14]	1b
	2	[128, 2]	[27, 27]	2a, 2b
			[13, 13]	5a, 5b
	3	[64, 2]	[13, 13]	3a, 3b
	4	[48, 2]	[13, 13]	4a, 4b
CD 4	0	[48, 1]	[14, 19]	1a
	1	[48, 1]	[14, 19]	1b
	2	[128, 2]	[27, 27]	2a, 2b
			[13, 13]	5a, 5b
	3	[64, 2]	[13, 13]	3a, 3b
	4	[48, 1]	[13, 13]	4a
	5	[48, 1]	[13, 13]	4b
Multi-core[2]	0	[48, 1]	[14, 19]	1a
	1	[48, 1]	[14, 14]	1b
	2	[64, 3]	[27, 27]	2a, 2b
	3	[64, 2]	[13, 13]	3a, 3b
	4	[96, 1]	[13, 13]	4a, 4b
	5	[64, 1]	[13, 13]	5a, 5b
Single-core[1]	0	[64, 9]	[8, 8]	1a, 1b
			[14, 27]	2a, 2b
			[13, 13]	3a, 3b
			[13, 13]	4a, 4b
			[13, 13]	5a, 5b

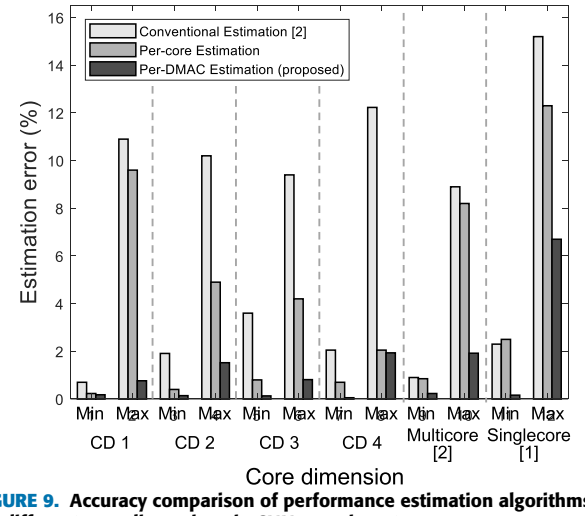


FIGURE 9. Accuracy comparison of performance estimation algorithms for different core dimensions in CNN experiments.

smaller estimation error than the other two performance estimation methods. The proposed per-DMAC estimation method exhibits an estimation error of <2% at all system

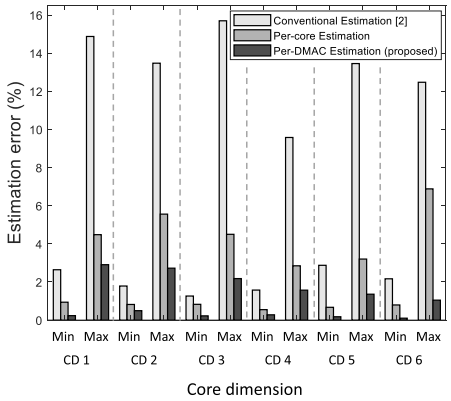


FIGURE 10. Comparison of different core dimensions. (a) Performance estimation results for communication bandwidth of 5 core dimensions and (b) performance gain of proposed core dimensions compared to the conventional multicore accelerator [2].

communication bandwidths for a multicore accelerator. The conventional estimation method [8] does not consider the number of active DMACs for each accelerator core and assumes that the system communication bandwidth is evenly assigned at all DMAC intervals, resulting in an error of up to 12%. For the proposed method, per-core estimation is performed by keeping track of the communication amount of the accelerator cores, and the performance estimation is slightly improved compared with the conventional estimation method [2], with a maximum error rate of 10%. However, because per-core estimation assumes that each accelerator core communicates using one DMAC, it is difficult to determine whether the performance was accurately estimated. The proposed per-DMAC estimation method tracks the communication amount and determines the number of active DMACs of each accelerator core, and it accurately estimates the communication bandwidth for each core of the DMA intervals and produces the best result. The estimation error is slightly increased in the result of the single-core accelerator. This was due to the underutilization of the on-chip bus, as the system communication bandwidth was limited not by the on-chip bus but by DRAM.

The performance estimation algorithms are also evaluated by experiments of wireless communication (e.g., 802.11n/ax). In these experiments, a core dimension denotes a set of parameters consisting of the number of antennas and modulation and coding scheme, instead of a set of tile sizes. Thus, different workloads are assigned to each of the three accelerator cores, i.e., fast Fourier transform engine, multiple-input-multiple-output detector and low-density parity-check decoder. As shown in Fig. 10, even in the case of wireless communication, the proposed algorithm predicts the performance more accurately than other algorithms. The proposed algorithm always exhibits estimation errors of <3%, whereas the conventional algorithm exhibits an error of 15.7%. Because the proposed algorithm can be applied in the same way even if the target application is changed, the results shown in Fig. 10 are not significantly different from the experimental results of the CNN.

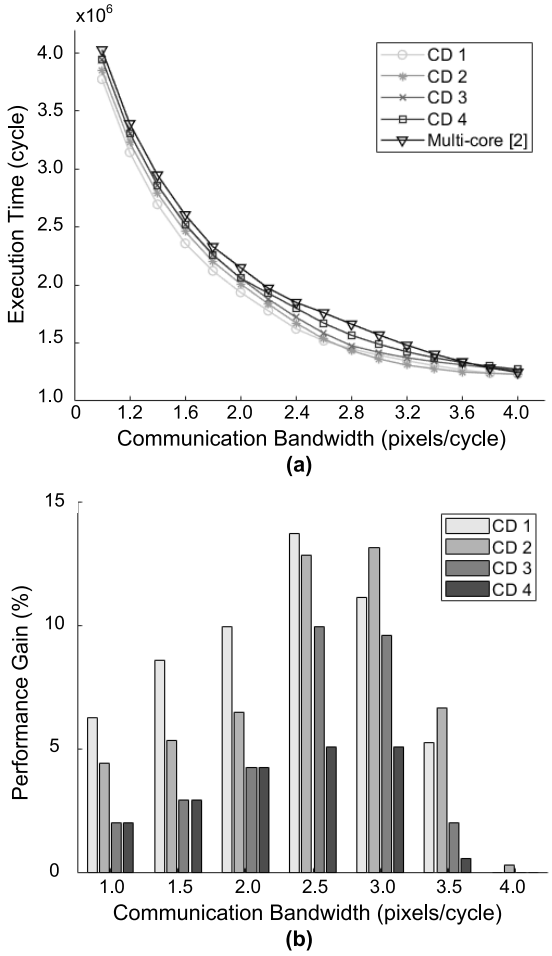


FIGURE 11. Block diagram of the on-chip bus implemented on the basis of the switch modules for each AXI channel.

C. ACCELERATOR OPTIMIZATION

The optimized multicore configurations which satisfy the specified hardware resource constraints using the core dimension optimization mentioned in [2] are explored by utilizing the proposed performance estimation algorithm.

We set the system communication bandwidth to be limited by the on-chip bus and verified through a simulation experiment that, when the system communication bandwidth was sufficient, the total execution time was determined by the computation time, as proposed in [2]. In other words, it can be found in Fig. 11(a) that all the proposed core dimensions have almost the same performance in the sufficient system communication bandwidth and have higher performance than those of the conventional core dimensions in the limited communication bandwidth (e.g., 4.0 pixels/cycle or less).

Fig. 11 shows the total execution time of the CNN accelerators for each system communication bandwidth (Fig. 11(a)) and the performance gain (Fig. 11(b)) compared with those of the conventional core dimension [2]. The CD1 using the proposed core dimension optimization had a maximum performance gain of 13.8% compared with that of the conventional core dimension [2] when the system communication bandwidth was 2.5 pixels/cycle. Whether the core dimension will show the best performance depends on the system communication bandwidth. As shown in Fig. 11(b), when the system communication bandwidth is 1.0-2.5 pixels/cycle, CD 1 shows excellent performance, and when the system communication bandwidth is 3.0-3.5 pixels/cycle, CD 2 shows the best performance. When the system communication bandwidth is more than 4.0 pixels/cycle, the execution time of all core dimensions, including the conventional core dimension [2], becomes dominant in the computation time and the performance is almost the same. However, that the execution time for each core dimension slightly differs by allowing a difference in computation time within a small range of error (within 2%) in the core dimension optimization.

Table 3 shows the performance gains shown in Fig. 11(b). The table shows the computation time (**Tp**), number of communication amounts (**Ni** + **Nw**), core execution time for each accelerator core of the conventional multicore dimension [2] and the proposed CD 1. This table shows the core execution time when the system communication bandwidths are 2.5 pixels/cycle (when the performance is the largest performance gain) and 4.0 pixels/cycle (when the performance is computation limited). The accelerator core should be executed before the total execution time becomes idle, and it is important to minimize the idle state by balancing the execution time of each core to achieve the best performance. When the system communication bandwidth is given as 2.5 pixels/cycle, each accelerator core of the conventional multicore dimensions [2] shows a maximum performance gap of 38.5%. This is because the number of communication amounts required to perform Core 3 and Core 4 is larger than that of the other cores. To solve this issue, we need to increase the communication bandwidth for each core assigned to Core 3 and Core 4 of the accelerator or to reduce the number of communication amounts by re-specifying the core dimensions. The proposed CD1 is one of the core dimensions resolved by increasing the communication bandwidth for each core assigned to Core 3 and Core 4 by reducing the number of accelerator cores. In the proposed CD1, Core 2 (TM = 64 and TC = 3) and Core 5 (TM = 64 and TC = 1), which finish execution relatively early in the multicore dimension [2], are integrated into Core 2. The previously assigned layers (layers 2 and 5) are sequentially executed in Core 2. The integrated Core 2 re-assigns the hardware resources used in the previous two cores and sets the core dimension (TM = 128 and TC = 2). As the number of cores decreases, the communication bandwidth assigned to each core increases and the gap in the execution time for each core decreases.

However, even in the proposed CD1, none of the execution times of each core completely converges, which is due to the use of the conventional core dimension optimization framework as a baseline. As shown in Fig. 8(b), we focus on maximizing the computational performance by dividing design space exploration into to step 1 (optimization of computation) and step 2 (optimization of memory).

TABLE 3. Computation time for each core of core dimensions, number of communication amount and core execution time for each system communication bandwidth.

Core Dimension	Core	Layers	Computation Time (Tp)	No. Communication Amount (Ni + Nw)	Core Execution Time (No. cycles x 1,000)	
					2.5 pixels/cycle	4.0 pixels/cycle
Multi-core[2]	0	1a	1,098,075	388,392	1,119	1,111
	1	1b	1,098,075	463,296	1,119	1,111
	2	2a, 2b	1,166,400	491,712	1,184	1,177
	3	3a, 3b	1,168,128	1,230,336	1,818	1,248
	4	4a, 4b	1,168,128	836,352	1,644	1,190
	5	5a, 5b	1,168,128	615,168	1,358	1,169
CD 1	0	1a	1,098,075	388,392	1,115	1,109
	1	1b	1,098,075	463,296	1,115	1,109
	2	2a, 2b	856,575	399,456	894	886
		5a, 5b	310,257	528,768	618	343
		Overall	1,166,832	928,224	1,512	1,229
	3	3a, 3b	1,168,128	1,230,336	1,598	1,200
	4	4a, 4b	1,168,128	836,352	1,397	1,174

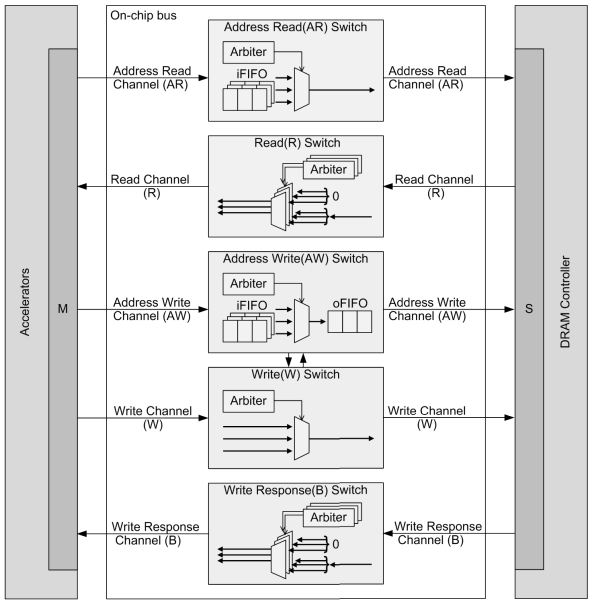


FIGURE 12. Block diagram of on-chip bus implemented based on switch modules for each AXI channel.

Specifically, the computational performance was maximized in step 1. In addition, unlike the conventional optimization assuming only computation limited case, the proposed core dimension optimization allows that the system communication bandwidth is insufficient to maximize the computational performance, and thus the candidate TE/TF may affect the execution time measurement.

VIII. HARDWARE IMPLEMENTATIONS

This section shows the implementation of the core dimension obtained through core dimension optimization using the proposed performance estimation algorithm and presents the implementation results. Multicore accelerators optimized in Section VII was implemented on Xilinx Zynq-7000 SoC ZC706 Evaluation Kit.

We designed accelerator cores excluding DMACs, with Vivado HLS (v. 2017.4). Each accelerator implementation includes a MAC array, on-chip buffers and an accelerator controller. The DMAC module is designed by using hand-written RTL codes and optimized to suit the AMBA AXI-4 interface [23]. Scatter-Gather DMACs generate source and destination addresses using address offsets, as described in Section VII. In addition, we set the data width of DMAC to 1 pixel or weight per beat for comparison with the simulation results. As mentioned above, the accelerator core is executed by starting or stopping the communication through the control information received from the microprocessor before the DMAC operates.

In addition, we implemented an on-chip bus as shown in Fig. 12 to connect multiple accelerator cores to off-chip DRAM. The on-chip bus is designed by using handwritten RTL codes based on the AMBA AXI-4 interface. Each submodule is dedicated to each AXI channel to ensure the channel independence of the AXI-4 interface. In addition, to efficiently handle the data requested by the DMACs of the multiple accelerator cores simultaneously, first-in-first-out (FIFO) data structures (i.e., input FIFO (iFIFO) and output FIFO (oFIFO)) are designed at the top of each submodule to minimize the bottleneck of the data transfer due to the on-chip bus. In other words, the on-chip bus receives and handles simultaneous requests from the DMACs as much as possible, minimizing the case wherein the DMAC requests are blocked by the contention occurring on the on-chip bus. The clock frequency of the microprocessor and DRAM device was set to 666 MHz and 500 MHz, respectively, and the clock frequency of the accelerator was set to 100 MHz, as was done in [1] and [2]. Moreover, the clock frequency of the communication was set to 100 MHz, which is the

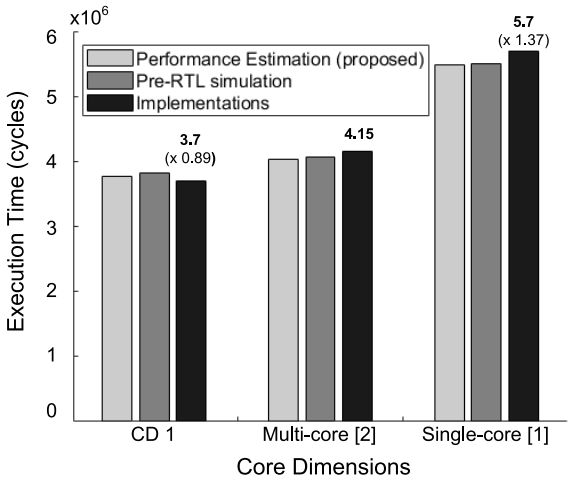


FIGURE 13. Results of the performance estimation algorithm, pre-RTL simulation, and implementations for the CD1, multi-core [2], and single-core [1] dimensions.

same as the frequency of the accelerator owing to implementation constraints. Therefore, when the system communication bandwidth is determined by the on-chip bus, rather than by DRAM, the system communication bandwidth is 1 pixel/cycle or 1 weight/cycle.

Fig. 13 shows the results of the performance estimation, simulation performance, and performance evaluated by the actual implementation of the proposed CD 1 and two conventional accelerator core dimensions (i.e., the single-core [1] and multicore [2] dimensions) listed in Table 2. This shows that the performance estimation results of each accelerator core dimension are close to the simulation results and the actual implementation performance results. In other words, the results of the proposed performance estimation algorithm show an error rate of up to 2.8% compared with the hardware implementation results. More importantly, in Fig. 13, the optimized core dimension (CD 1) shows a performance gain of 31.2% compared with that of the conventional single-core dimensions [1] and 10.8% compared with that of the multicore dimensions [2].

IX. CONCLUSION

In this paper, we propose a novel performance estimation algorithm which accurately estimates the performance of a multicore accelerator. The proposed algorithm predicts the communication amount of each DMA interval by considering the dynamic communication bandwidth on the basis of the number of active DMACs. The pre-RTL simulation results reveal that the proposed per-DMAC performance estimation algorithm shows the estimation error of less than 2.8%, regardless of the system communication bandwidth. The proposed algorithm is also verified against the hardware implementation results. In addition, it is shown that, when the core dimension is optimized by using the proposed algorithm, the resulting performance gain amounts to 10.8% and 31.2%, when compared with the conventional multicore accelerator and single-core accelerator, respectively. Note that the proposed algorithm is generally applicable to accelerators with a wide range of applications, and this study includes experiments on accelerators for CNNs and wireless communications.

REFERENCES

[1] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp.*, Feb. 2015, pp. 161–170.

[2] Y. Shen, M. Ferdman, and P. Milder, "Maximizing CNN accelerator efficiency through resource partitioning," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 535–547.

[3] J. Shen, Y. Qiao, Y. Huang, M. Wen, and C. Zhang, "Towards a multiarray architecture for accelerating large-scale matrix multiplication on FPGAs," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2018, pp. 1–5.

[4] M. Sinha, G. S. Harsha, P. Bhattacharyya, and S. Deb, "Design space optimization of shared memory architecture in accelerator-rich systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–31, 2021.

[5] S. Kim, C. Im, and S. Ha, "Schedule-aware performance estimation of communication architecture for efficient design space exploration," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 13, no. 5, pp. 19–24, May 2005.

[6] Y. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+: Fast and accurate performance estimation for FPGA HLS," in *Proc. Int. Conf. Comput. Aided Des.*, Nov. 2017, pp. 691–698.

[7] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Performance modeling for CNN inference accelerators on FPGA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 4, pp. 843–856, Apr. 2020.

[8] Y. Zhao, C. Li, Y. Wang, P. Xu, Y. Zhang, and Y. Lin, "DNN-chip predictor: An analytical performance predictor for DNN accelerators with various dataflows and hardware architectures," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 1593–1597.

[9] A. X. M. Chang and E. Culurciello, "Hardware accelerators for recurrent neural networks on FPGA," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2017, pp. 1–4.

[10] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 11, pp. 2072–2085, Nov. 2019.

[11] T. Abtahi, C. Shea, A. Kulkarni, and T. Mohsenin, "Accelerating convolutional neural network with FFT on embedded hardware," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 9, pp. 1737–1749, Sep. 2018.

[12] Y. Ma, Y. Cao, S. Vrudhula, and J.-S. Seo, "Optimizing the convolution operation to accelerate deep neural networks on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 7, pp. 1354–1367, Jul. 2018.

[13] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[14] P. Meloni, G. Deriu, F. Conti, I. Loi, L. Raffo, and L. Benini, "A high-efficiency runtime reconfigurable IP for CNN acceleration on a mid-range all-programmable SoC," in *Proc. Int. Conf. ReConFigurable Comput. FPGAs (ReConFig)*, Nov. 2016, pp. 1–8.

[15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Nov. 2017.

[16] S. Hessel, D. Szczesny, F. Bruns, A. Bilgic, and J. Hausner, "Architectural analysis of a smart DMA controller for protocol stack acceleration in LTE terminals," in *Proc. IEEE Veh. Technol. Conf. (VTC)*, Sep. 2010, pp. 1–5.

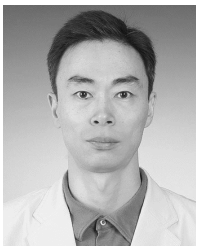
[17] S. Sirowy, Y. Wu, S. Lonardi, and F. Vahid, "Clock-frequency assignment for multiple clock domain systems-on-a-chip," in *Proc. Design, Autom. Test Eur. Conf. Exhib.*, Apr. 2007, pp. 1–6.

[18] A. Mehrabian, M. Miscuglio, Y. Alkabani, V. J. Sorger, and T. El-Ghazawi, "A winograd-based integrated photonics accelerator for convolutional neural networks," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 1, Jan. 2020, Art. no. 6100312.

[19] S. Shivapakash, H. Jain, O. Hellwich, and F. Gerfers, "A power efficiency enhancements of a multi-bit accelerator for memory prohibitive deep neural networks," *IEEE Open J. Circuits Syst.*, vol. 2, pp. 156–169, 2021.
[20] X. Wei, Y. Liang, and J. Cong, "Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management," in *Proc. Design Automat. Conf. (DAC)*, Jun. 2019, pp. 1–6.
[21] S. Kim, J. Wang, Y. Seo, S. Lee, Y. Park, S. Park, and C. S. Park, "Transaction-level model simulator for communication-limited accelerators," 2020, *arXiv:2007.14897*.
[22] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A cycle accurate memory system simulator," *IEEE Comput. Archit. Lett.*, vol. 10, no. 1, pp. 16–19, Jan. 2011.
[23] *AMBA AXI ACE Protocol Specification, AXI3, AXI4, AXI4-Lite, ACE ACE-Lite*. Cambridge, U.K.: ARM Infocenter, 2011.
[24] M. Sinha, G. S. Harsha, P. Bhattacharyya, and S. Deb, "Design space optimization of shared memory architecture in accelerator-rich systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 4, pp. 1–31, Apr. 2021, doi: 10.1145/3446001.

SUNWOO KIM received the B.S. degree in electronics engineering from Konkuk University, Seoul, South Korea, in 2014, where he is currently pursuing the Ph.D. degree in electronics engineering. His research interests include modeling and simulation for SoC architecture and algorithm-architecture co-design for hardware accelerators.

YOUNGHO SEO received the B.S. degree in electronics engineering from Konkuk University, Seoul, South Korea, in 2019, where he is currently pursuing the M.S. degree in electronics engineering. His research interests include modeling and simulation of SoC and hardware accelerators.



SUNGKYUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electronics engineering from Seoul National University, South Korea, in 2002. From 2002 to 2004, he was a Senior Engineer with Samsung Electronics, where he worked on the development of system-level simulators for cellular standards. From 2004 to 2006, he was a Senior Member of Research Staff with the Electronics and Telecommunications Research Institute (ETRI), where he worked on fiber-optic front-end IC design. From 2006 to 2009, he was a Senior Staff Hardware Designer with Ericsson Inc., where he worked on the design and modeling of multi-standard RF transceivers and clocking circuits. In 2009, he joined the Faculty of the Department of Electronics Engineering, Pusan National University, South Korea, where he is currently a Professor. His research interests include design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G.

CHESTER SUNGCHUNG PARK (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 2006. From 2006 to 2007, he was with Samsung Electronics, Giheung, South Korea. From 2007 to 2013, he was with Ericsson Research, USA, as a Senior Engineer. Since 2013, he has been with the Department of Electronics Engineering, Konkuk University, South Korea, as an Associate Professor, where he is currently working on the design and modeling of SoC, hardware accelerators, and virtual platforms for neural networks and 5G. His research interests include SoC architecture design for artificial intelligence, processing in memory, and wireless communication.