Optimization of Communication Schemes for DMA-Controlled Accelerators

JOOHO WANG, SUNGKYUNG PARK, (Senior Member, IEEE), AND CHESTER SUNGCHUNG PARK, (Senior Member, IEEE)

1Department of Electrical and Electronics Engineering, Konkuk University, Seoul 05029, South Korea
2Department 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 The hardware accelerator controlled by direct memory access (DMA) is greatly influenced by the communication bandwidth from/to DRAM through on-chip buses. This paper proposes a novel performance estimation algorithm to optimize the communication schemes (CSs), which are defined by the number of direct memory access controllers (DMACs) and the bank allocation of DRAM. In order to facilitate the optimization of CSs, a communication primitive (CP) is defined by the bank allocation and the set of activated DMACs. By using the communication bandwidths of CPs obtained from prior full-system simulations, the proposed performance estimation algorithm can predict the communication performance of CSs more accurately, compared with the conventional performance estimation algorithms. When it is applied to convolutional neural networks (CNNs) and wireless communications (LDPC-coded MIMO-OFDM), the estimation error is measured to be no more than 6.4% and 5%, respectively. In addition, compared with the conventional simulation-based approaches, the proposed estimation algorithm provides a speedup of two orders of magnitudes. The proposed performance estimation algorithm is used to optimize the CS of the CNNs and explore a design space characterized by bank interleaving, outstanding transactions, layer shape, tile size, and hardware frequency. It is shown that the optimized CS improves communication performance by up to 68% for the third convolutional layers of AlexNet and 60% for the MIMO of LDPC-coded MIMO-OFDM. In addition, the DRAM latency is minimized by setting the bank interleaving to the number of outstanding transactions. Moreover, the simulation results show that the optimum CS depends on the application. It is also shown that the use of an extra DMAC does not necessarily improve the communication performance.

INDEX TERMS Convolutional neural networks, direct memory access, hardware accelerator, off-chip DRAM, on-chip communication, wireless communications.

I. INTRODUCTION
Recently hardware accelerators have become one of the most effective solutions in many areas such as machine learning. Hardware accelerators are often equipped with local memories controlled by direct memory access controllers (DMACs) [1]–[5]. Most of the DMA-controlled accelerators use the well-known double buffering to improve the performance by overlapping computation with communication [6]–[9]. If the hardware accelerator as a standalone IP is connected to off-chip memory (e.g., DRAM) through an on-chip bus (e.g., AXI crossbar), the communication bandwidth, which is defined as the number of data items per cycle transferred from/to accelerator, tends to be limited by either DRAM latency or bus protocol overhead [6]–[11].

This paper deals with the optimization of communication schemes (CSs) for DMA-controlled accelerators, which are characterized by the number of direct memory access controllers (DMACs) and the bank allocation of DRAM. In detail, as shown in [6]–[13], one or more DMACs of an accelerator access DRAM subsystem through the on-chip bus as bus masters. Moreover, the data layout of DRAM subsystem depends on how each data type is allocated multiple banks. In addition, in order to speed up the evaluation of
CSs, we newly propose a performance estimation algorithm based on communication primitives (CPs), each of which is defined by the number of activated DMACs and per-DMAC bank allocations. Both the CS optimization and the performance estimation algorithm are generally applicable to any DMA-controlled accelerators that access off-chip memory through on-chip buses [6]–[13], which include some of the commercial SoC platforms such as Xilinx ZYNQ [14] and NVIDIA NVDLA [15].

We summarize the contributions of this study as follows. First, the proposed performance estimation algorithm can predict the communication performance of CSs more accurately since it takes into account both bus protocol overhead and DRAM latency by making use of the full-system simulations for evaluating the communication bandwidths of the constituent CPs compared with the CAG-based bus performance estimation [16] and the statistics-based DRAM latency estimation [17]. The proposed algorithm reduces the performance estimation error down to 5% whereas the conventional algorithms in [16] and [17] experience the estimation error of 60% and 21%, respectively. Moreover, it is verified that the proposed algorithm is generally applicable to estimate the performance of any DMA-controlled accelerators. For example, the estimation error is measured to be no more than 6.4% for the convolutional layers of CNNs and no more than 5% for the LDPC-coded MIMO-OFDM. Second, the proposed performance estimation algorithm provides a speedup of two orders of magnitudes compared with the conventional simulation-based approach. For example, according to our experiments, it takes about a few hours to evaluate a single CS with hundreds of different combinations of tile sizes and accelerator frequencies. In contrast, in order to minimize the simulation time and maintain the estimation accuracy, the proposed algorithm constrains the use of full-system simulator to evaluating the communication bandwidths of the constituent CPs. Since a few tens of CPs suffice to express most of the CSs of interest, the extra simulation time to obtain the communication bandwidths of CPs becomes negligible, particularly, in the case of a broad design space (i.e., a space of hundreds of design points). Namely, once the communication bandwidths of CPs are obtained from prior simulations, it is possible to estimate the performance of any CS on a per DMA interval basis, importantly, without any additional simulations, since each CS can be expressed as a CP set.

The remainder of this paper is organized as follows. Section II reviews some of the related works and summarize our contributions. In Section III, the system considered in this paper is introduced. In Section IV, the DRAM data layout and access pattern assumed in this paper are introduced together with bank interleaving. Moreover, the impact of outstanding transactions are also introduced. In Section V, a CP of a DMA-controlled accelerator is defined. Based on this, Section VI shows that a CS of a DMA-controlled accelerator is expressed as a set of CPs. In Section VII, a primitive-based performance estimation algorithm is proposed for estimating the performance of CSs. Section VIII evaluates the estimation accuracy and simulation time of the proposed estimation algorithm. In addition, this section compares the communication performance of several CSs. Finally, Section IX concludes this paper.

II. RELATED WORKS

There have been several reports on the optimization of the bank allocation of DRAM [18], [19]. Most of the proposed bank allocations focus on balancing interference mitigation and bank-level parallelism. For example, in [18], DRAM banks are dynamically partitioned according to the application profiling (e.g., memory-intensive vs. non-intensive). In [19], a locality-aware bank partitioning (LABP) is proposed to improve the dynamic bank partitioning by mitigating the inference caused by non-intensive applications, for example, by separating their banks from those from memory-intensive high row-buffer locality applications. However, the proposed bank allocations are targeted for general purpose processors running different applications. The optimization of bank allocation for application-specific hardware accelerators (in particular, DMA-controlled accelerators) has not been considered. Moreover, it is pointed out in [7] that the communication bandwidth is affected by the bus protocol overhead as well as the DRAM bank allocation.

There have been many reports on DMA-controlled hardware accelerators. In general, the communication performance improves with the number of DMACs at the cost of additional hardware complexity. In [12], a hardware accelerator is equipped with two separate DMACs, one for data input and one for data output. In [13], the use of multiple read DMACs improves the communication performance of a hardware accelerator. However, none of these evaluates the performance gain of adding extra DMACs. In fact, the communication performance does not necessarily improve with the number of DMACs, as will be shown later in this paper. An extra DMAC may end up increasing the hardware complexity without providing any significant performance gain. Therefore, it is important to decide the optimum number of DMACs of a hardware accelerator.

When it comes to the implementation of hardware accelerators, there are several hardware constraints [20], [21]. For example, in [20], it is shown that on-chip interconnects between hardware components often limit the maximum achievable performance of the entire system. In addition, it is also shown that the performance of a hardware accelerator is often limited by DRAM latency, which tends to be degraded by locality failures (e.g., row buffer misses). In [21], it is shown that the processor core control overhead for synchronizing hardware accelerators may affect the accelerator performance severely. The system considered in this paper includes that several hardware components and schemes are assumed in order to overcome the hardware constraints mentioned in [20], [21]. For example, in order to overcome the interconnect hardware constraint, the hardware accelerator considered in our work is assumed to utilize an AXI-based on-chip crossbar bus. In addition, it is also assumed that each
of the DMACs has a capability of supporting multiple outstanding. This helps improve the communication bandwidth of a hardware accelerator, especially in the communication-limited case. Moreover, in order to alleviate the DRAM latency, the proposed performance estimation algorithm can be used to optimize the DRAM data layout (bank allocation) and the DRAM access pattern (bank interleaving). Lastly, the hardware accelerator is based on the assumption that a processor core is responsible for starting and stopping the execution of the DMACs appropriately. Thus, the communication performance tends to be dependent on the processor core hardware constraints. For the reason, the proposed performance estimation algorithm takes into account the impact of processor core constraints on the communication performance. However, according to the measurements on Xilinx ZYNQ [14], the processor core control overhead is usually too small to influence the accelerator performance, as will be shown later in this paper.

When it comes to the estimation of communication performance, there are two different approaches: static analysis-based approach and simulation-based approach. Static analysis-based approach tends to be faster than simulation-based approach, but the estimation accuracy may not be high enough to drive the design of communication architectures. Especially, static analysis-based approach lacks the ability to capture the dynamic nature of the communication bandwidth accurately. In order to improve the accuracy, static analysis based approach is often combined with a set of traces extracted from simulations. In [16], a hybrid trace-based performance estimation algorithm is proposed to estimate the performance of bus-based communication architectures using a communication analysis graph (CAG). In [22], a simulation-based performance estimation is proposed based on the observation of the actual traffic tracking of the application of each core (e.g., bus master). In [23], in order to estimate the performance of bus-based communication architectures, the static analysis based on a modified queueing model is incorporated into the schedule-aware performance estimation. In [7], a revised roofline model is proposed to estimate the performance of a DMA-controlled accelerator, taking into account the impact of DRAM latency. Reference [17] proposes to estimate the DRAM latency based on the statistics of different access conditions (e.g., row buffer hit). However, none of these models the communication bandwidth of a DMA-controlled accelerator, in particular, from/to DRAM subsystem through on-chip bus. Moreover, none of these takes into account the bank allocation of DRAM considered in this paper.

On the other hand, simulation-based approach is accurate enough to capture dynamic communication bandwidth. Thus, most of the reports on bank allocation rely on the performance evaluation using full-system simulators together with DRAM simulators. For example, in [18], the proposed bank allocations are evaluated using gem5 together with DRAMSim2. In [24], a full-system simulator models the communication bandwidth of a DMA-controlled accelerator from/to DRAM subsystem through on-chip bus on a transaction level. However, simulation-based approach may often be too time-consuming to be used for exploring a broad design space of communication architectures.

We summarize the novelty of this study as follows:

- We propose a novel performance estimation algorithm to optimize the CS, which is defined by the number of DMACs and the bank allocation of DRAM. In order to facilitate the optimization of CSs, we newly propose to define a CP by the number of activated DMACs and the per-DMAC bank allocation such that each CS can be expressed as a set of CPs. The proposed primitive-based performance estimation algorithm evaluates the performance of a CS based on the communication bandwidths of the constituent CPs, for the first time, to the best of our knowledge. Moreover, in order to improve both estimation accuracy and simulation speed, the proposed performance estimation algorithm newly incorporates the simulation-based approach into the schedule-aware performance estimation algorithm in [23]. Lastly, the proposed performance estimation algorithm firstly considers the performance impact of both DRAM latency and bus protocol overhead on a DMA-controlled accelerator.

- Using the proposed performance estimation algorithm, a variety of CSs of a DMA-controlled accelerator are newly evaluated. To the best of our knowledge, the communication performance of various bank allocations is firstly compared, taking into account the performance impact of both DRAM latency and bus protocol overhead (in the context of DMA-controlled accelerator). The experimental results show that the optimizations of CS are not straightforward, and, more importantly, they are heavily dependent on the application. For example, in the case of CNNs, it is shown that the optimum CS varies with the layer shape and the tile sizes. Moreover, there are a couple of interesting observations on the optimizations of CS. For example, it is shown that the use of an extra DMAC does not necessarily improve the performance unless the bank allocation is carefully chosen. It is also shown that the optimality of bank allocation may depend on the number of the available DMACs.

### III. SYSTEM UNDER CONSIDERATION

The system assumed in this paper consists of a hardware accelerator, a single processor core, a DRAM subsystem and an on-chip bus. The hardware accelerator is assumed to be equipped with multiple DMACs, as shown in [6]–[8]. In detail, each of the DMACs serves as a bus master and accesses the DRAM subsystem through the on-chip bus. The hardware accelerator is assumed to be equipped with either one ([1], [8], [9]) or multiple DMACs ([2], [5]–[7]) and each of the DMACs accesses the DRAM subsystem as a bus master. For example, Figure 1 (a) shows a CNN
operates at the clock frequency of 533 MHz in [31], it is assumed to operate at the clock frequency of more than 200 MHz, it takes about 80 cycles per DMAC to start the DMAC execution. Lastly, the hardware accelerator is responsible for setting the source/destination addresses appropriately [25]–[27]. Recalling that the processor core is the processor core makes it possible to reconfigure the hardware accelerator according to the bank allocations and number of DMACs. Since a hardware accelerator is usually designed as a standalone IP block, a standardized interface may ease the integration into the system [1], [2], [6]–[9]. The AMBA AXI4 interface, the standardized interface used in [28], is assumed in this work, as illustrated in Figure 1 (a). The single processor core starts and stops the DMACs [29] by reading from or writing to their control registers. The measurement results show that, with the core frequency of (667 MHz) and the DMAC frequency of (200 MHz), it takes about 80 cycles per DMAC to start the DMAC execution. Lastly, the hardware accelerator is assumed to operate at the clock frequency of more than 1 GHz, e.g., 1.17 GHz in [30]. Recalling that DDR3 often operates at the clock frequency of 533 MHz in [31], it is reasonable to assume that the hardware accelerator operates twice as fast as the off-chip DRAM.

Figure 1 (b) exemplifies a processing pass of a convolutional layer with three DMACs for three data types. A processing pass starts by loading the local memories with the input feature maps and filters of a predefined tile size from DRAM through RI-DMAC and RW-DMAC, respectively. Subsequently, the processing element (PE) array takes input pixels and weights, and, after the computation, return partial sums back to the local memories. Upon completion of the convolution computation over a predefined tile size, the resulting output feature maps are stored into DRAM through the WO-DMAC, which completes a processing pass. Assuming the use of double buffering, the duration of a processing pass is determined by the maximum between the communication time (i.e., the time required for load/store operations) and the computation time (i.e., the time required for PE computations) [32]–[37]. Depending on which is longer, it is referred to as communication-limited or computation-limited. For example, in the third convolutional layer of AlexNet [38], the layer shape is set as $M = 384, C = 256, E = 13, F = 13, R = 3,$ and $S = 3$ and the tile sizes are set as $TM = 64, TC = 2, TE = 13,$ and $TF = 13$. In this case, assuming the communication bandwidth of one data transfer per cycle, the 2816 cycles for computation time is much larger than the 1521 cycles for computation time (as will be shown in the later section). There have been many reports about communication-limited hardware accelerators. For example, the roof-line model in [6] calculates the communication time based on the communication bandwidth and the communication amount. It is pointed out in [7] that the communication time can be calculated more accurately by introducing the empirically-chosen communication bandwidth. Moreover, in [33], it is pointed out that, if neither the tile size of weights nor the pixels of the convolutional layer can be fully buffered to on-chip memory, the accelerator may become communication-limited. Most of these reports propose to improve the performance of a communication-limited hardware accelerator by using several loop transformation techniques. In this paper, we focus on the communication-limited case where the accelerator performance is determined by the communication bandwidth. DMACs communication time can be longer than PE array computation time, and it is defined as communication-limited in a processing pass. It is assumed that each of the DMACs is initiated by the processor core, one at a time. Moreover, the communication amount (i.e., the number of pixels or weights) depends on the tile size and thus it generally varies with the data type. Therefore, the DMACs may not complete their communication task at the same time in a processing pass. Depending on the set of activated DMACs, it is possible to partition a processing pass into multiple sub-passes, each of which is referred to as a DMA interval. For instance, a processing pass is partitioned into five DMA intervals in Figure 1 (b) assuming the initiation order of WO-DMAC, RI-DMAC and RW-DMAC: a single DMAC (WO-DMAC)
runs in the DMA interval 0, two DMACs (WO-DMAC and RI-DMAC) run in the DMA interval 1, a single DMAC (RI-DMAC) runs in the DMA interval 2 and so on. Note that the duration of a DMA interval is determined by either the DMA set time (i.e., when a DMAC is initiated running) or the DMA transfer time (i.e., when it completes the communication task). For example, the duration of the DMA interval 0 is determined by the DMA set time of RI-DMAC, whereas that of the DMA interval 1 is by the DMA transfer time of WO-DMAC. In addition, the DMA transfer time is determined by its communication bandwidth. Thus it can be concluded that, given the DMA set time, the duration of a DMA interval is determined by the communication bandwidth of the activated DMACs.

IV. COMMUNICATION PARAMETERS

In this section, several parameters are introduced to define the CP and CS considered in this paper. For example, the DRAM-related communication parameters include the bank allocation and the bank interleaving, which characterize the DRAM data layout and access pattern, respectively. In addition, the DMAC-related communication parameters include the number of bus masters and the number of outstanding transactions, which characterize the bus interface architecture. The DRAM-related communication parameters do through DRAM latency while the DMAC-related communication parameters affect the communication bandwidth through bus protocol overhead latency.

A. BANK INTERLEAVING

Each DRAM bank makes use of its row buffer to cache the most recently accessed memory row of the bank. In detail, when a row of the bank is accessed, the entire row of the bank (also known as page) is opened and the data stored in the row are transferred into the row buffer (activate). While a row is active in the row buffer, any number of column accesses (read or write) may be performed, typically within one cycle. After completing the required column accesses, the cached row must be written back to the memory array by an explicit operation (precharge), which closes the page and prepares the bank for a subsequent memory access. For example, Figure 2 (a) illustrates the DRAM bank state with DMAC0 reading from the same row of a bank, B0, assuming the maximum bus burst length (note that the burst length is defined as the number of data transfers for a burst) of 8 and up to six outstanding transactions. The figure includes some of the AXI channels (AR, R, AW, and W) [28]. Once the first bus request from DMAC0 arrives at DRAM, it opens the page by executing activate (ACT). Subsequently, the first five bus requests are served by executing five consecutive reads (R), i.e., one read per bus request, and then the page is closed by executing precharge (PRE). Note that the remaining bus requests including the sixth bus request cannot be served before the (first) page close (although they are addressed to the same row). In other words, every page open entails five consecutive reads. This can be explained by the 4-time close policy [39]. In addition, Figure 2 (b) illustrates the DRAM bank state with an additional DMAC (DMAC1) reading from another bank, B1. Each bank is accessed independently of the other bank with each executing five consecutive reads per page open.

In general, the more contiguous the data layout are, the fewer page opens the outstanding bus requests result in [40]. In particular, the outstanding bus requests may result

---

**FIGURE 2.** Communication activities of a couple of CPs: (a) 1R, (b) 1R2R, (c) 3R2R, (d) 1W2R.
in even a single page open if they arrive before the page close. In addition, it is possible to adjust the access pattern to further reduce the number of pages opens. Therefore, it can be concluded that it is possible to improve the DRAM latency by either making the DRAM data layout more contiguous or adjusting the access patterns.

Figure 3 exemplifies the data layout of the feature maps assumed in this work. For simpler illustration, four input feature maps, four rows per input feature map, two DRAM banks and four rows per DRAM page are assumed. Within a bank, the data layout assigns the pixels of a feature map to the DRAM locations in a row-major order. Recalling that it is more efficient to occupy a bank with as many reads or writes as possible (rather than activate or precharge) from the viewpoint of DRAM usage [41], it is reasonable to determine data layout in row-major order. Thus it follows that the more reads or writes per page open are executed, the more efficient the DRAM usage is and thus the lower the DRAM latency is. In addition, as shown in this figure, in the presence of multiple DRAM banks, they are evenly spread over all the banks so that the same bank allocation can be seen by all the processing passes.

Given a specific data layout, the communication bandwidth is determined by the access pattern. If a DMAC is allocated multiple banks, the access pattern is determined solely by the bank interleaving, which defines how often the DMAC switches the bank to access. It is not straightforward to decide the optimum bank interleaving. Generally, it is desirable to steer the DMAC to different banks in order to mitigate bank collision and exploit bank-level parallelism. Figure 4 compares the impact of bank interleaving on the communication bandwidth, assuming two DMACs supporting two outstanding bus requests in two DRAM banks. In addition, each DMAC is assumed to be allocated to two banks. The figure illustrates the DRAM bank state with two DMACs (DMAC0, DMAC1) reading from different rows of two banks (B0, B1), assuming the maximum bus burst length of 8 and up to two outstanding transactions. In Figure 4 (a), each DMAC switches the bank to access every bus request (bank interleaving per bus request). For example, the first bus request is addressed to B0, the second bus request is addressed to B1, and so on. Therefore, each bank ends up being addressed by the bus requests from different DMACs, in other words, alternating the requesting DMAC every bus request. This causes each bank to open (and close) a page every bus request, thereby executing a single read per page open. For example, each bank (e.g., B0) is addressed by the first bus request from DMAC0 and subsequently the first bus request from DMAC1. Recalling that the two bus requests are addressed to different rows of B0, the page opened by the former bus request needs to be closed before the latter bus request can be served. In contrast, in Figure 4 (b), by increasing the bank interleaving to two, the outstanding bus requests can be executed with a single page open. Furthermore, the bank-level parallelism can be exploited more efficiently than with the bank interleaving larger than two, e.g., in Figure 4 (c). As obviously shown here, it is generally better to base the bank interleaving on the number of outstanding transactions, which is assumed throughout this paper unless mentioned otherwise.

B. OUTSTANDING TRANSACTIONS

The AXI protocol assumes that a bus master has a capability of supporting multiple outstanding transactions [28]. In other words, a bus master can issue a predefined number of new bus requests before completing the already-issued ones. This may help improve the communication bandwidth especially in the communication-limited case.

Figure 5 shows how the bus requests issued by one or two DMACs are executed in DRAM. In Figure 5 (a), assuming a single master supporting two outstanding bus requests, the DRAM bank is not fully populated by DRAM commands. This can be seen as being in the bus-limited case in the sense that the next outstanding bus requests arrive at the DRAM device no earlier than the corresponding bank being idle. However, in Figure 5 (b), assuming the support of four outstanding bus requests, the DRAM bank remains busy most of the time, improving the communication bandwidth by 63%. It can be concluded that the communication bandwidth largely increases with the number of outstanding transactions since (some of) the next outstanding bus requests arrive earlier than the DRAM bank being idle (even earlier than the page close).
In the presence of an extra read DMAC, it is possible to run one DMAC for input feature maps (RI-DMAC) and the other DMAC for filters (RW-DMAC) at the same time in CNNs. Such a three-master bus interface increases the effective number of outstanding bus requests, i.e., those seen by the DRAM subsystem. In general, the more bus masters the accelerator runs at the same time, the busier the DRAM bank can become and thus the more efficient the DRAM usage can be. Thus, the use of extra DMACs may improve the communication bandwidth especially in the bus-limited case. For example, the comparison of Figure 5 (c) with Figure 5 (b) shows that the simultaneous use of two read DMACs improves the communication bandwidth by 25%. Roughly speaking, this is equivalent to doubling the number of outstanding transactions except that it results in an additional page open (since the addresses of the two masters are generally not contiguous). In addition, it is worth mentioning that this bandwidth gain comes at the cost of additional hardware for the extra DMACs.

V. COMMUNICATION PRIMITIVES

A CP is defined by the number of DMACs and per-DMAC bank allocations. Figure 6 shows some of the CPs with three banks and up to three DMACs. In particular, the number of banks allocated to a DMAC is determined according to the bank allocation and is assumed to be 1, 2, or 3, as shown in Figure 6. Each per-DMAC bank allocation is defined by a bank map, the 3-tuple (B2, B1, B0) whose element represents whether the DMAC is allocated the corresponding bank or not. For example, replacing a 3-tuple by a decimal number, 1W refers to the CP with the write DMAC (W) being allocated B0 (i.e., (0, 0, 1)). It is said to be equivalent to 4W, the CP with the write DMAC (W) being allocated B2 (i.e., (1, 0, 0)) since both of them have a single DMAC and the same bank allocation. Similarly, 4W2R1R refers to the CP with the write DMAC (W) being allocated B2 (i.e., (1, 0, 0)), the first read DMAC (R0) being allocated B0 (i.e., (0, 0, 1)) and the second read DMAC (R1) being allocated B1 (i.e., (0, 1, 0)). Moreover, it is equivalent to 2W4R1R since all of them have three DMACs and the same bank allocation.

Figure 2 shows the communication activities of a couple of CPs assuming the maximum bus burst length of 8 and up to six outstanding transactions. For example, a comparison of Figure 2 (a) with (b) shows that the simultaneous use of two read DMACs improves the communication bandwidth by 64%. In general, the more DMACs the hardware accelerator in Figure 1 (a) uses currently, the busier the DRAM bank can become and thus the more efficient the DRAM usage can be. However, as shown in Figure 2 (c), when two different DMACs simultaneously access the B1, communication bandwidth is reduced owing to a bank collision. In other words, the use of extra DMACs does not always lead to improvement of the communication bandwidth. Lastly, in Figure 2 (d), it is shown that the communication bandwidth of a read DMAC is independent of that of a write DMAC.
in case of no bank collision since the AXI interface support separate read and write channels. Therefore, in principle, the communication bandwidth of 1W2R may exceed the unity (i.e., more than one data items per cycle). Figure 7 shows the communication bandwidths of several CPs with respect to accelerator frequency. Here the hardware accelerator frequency is normalized to DRAM subsystem frequency and it is assumed to be as much as twice the DRAM subsystem frequency [30]. In Figure 7, the hardware accelerator frequency is assumed to range from a minimum of 125 MHz to a maximum of 1000 MHz [42]–[44]. In addition, the DRAM frequency is assumed to be fixed at 500 MHz [45]. Therefore, the accelerator frequencies of 125 MHz and 1000 MHz corresponds to 0.25× frequency and 2× frequency, respectively. When the accelerator frequency is sufficiently low, the DRAM runs fast enough to serve the bus requests from the accelerator promptly and thus the communication bandwidth is bus-limited. However, as the accelerator frequency increases, the DRAM runs relatively more slowly (thereby failing to serve the bus requests promptly) and thus the communication bandwidth is DRAM-limited, causing the communication bandwidth to drop down. For example, in the case of 1R, the accelerator performance is bus-limited at 0.25× frequency (pointed by A) whereas it is DRAM-limited at 2× frequency (pointed by B). Figure 8 compares the DRAM bank state for the two accelerator frequencies. It is shown that, at 0.25× frequency, the bus requests can be served as soon as they arrive at DRAM and thus the communication bandwidth saturates to the unity (i.e., bus-limited), as shown in Figure 8 (a). On the other hand, at 2× frequency, the DRAM bank is fully populated with DRAM commands (i.e., never being idle) and therefore the communication bandwidth is limited by the DRAM page opens (i.e., DRAM-limited), dropping down to 0.58, as shown in Figure 8 (b). In the low frequency regime, the hardware accelerator tends to be limited by bus protocol overhead and thus the communication bandwidth tends to be equally split among DMACs, as assumed in the static analysis of [16]. In particular, in case that on-chip bus is fully utilized, the communication bandwidth of each DMAC is given as

\[ BW = \frac{1}{N\_DMAC} \]  

(1)

where \( N\_DMAC \) denotes the number of DMACs. Here note that the communication bandwidth of WO-DMAC is
independent of that of either RI-DMAC or RW-DMAC since the AXI interface supports the physically separate read and write channels. For example, in the case of 4W2R1R, the communication bandwidth of a read DMAC approaches 0.5 whereas that of a write DMAC does 1.0, as shown in the figure. On the other hand, in the high frequency regime, this argument does not hold any more. More specifically, the hardware accelerator may be limited by DRAM latency (not bus protocol overhead). For example, the communication bandwidth of 4W2R1R approaches 0.34 for a read DMAC and 0.19 for a write DMAC as the accelerator frequency increases. In this case, the communication bandwidth is heavily dependent on the DRAM condition, e.g., row buffer hit/miss, as pointed in [17]. For example, it is shown in the figure that, in the case of 1R and 1R1R, the total communication bandwidth approaches 0.58, as predicted by the statistics-based DRAM latency estimation in [17]. However, it is not generally applicable to the case with multiple DMACs and multiple DRAM banks (as will be shown in the later section). Furthermore, the DRAM-limited communication bandwidth tends to deviate from the actual communication bandwidth in the low frequency regime, as shown in Figure 7.

VI. COMMUNICATION SCHEMES

In this section, we define a CS based on the number of DMACs and the per-type bank allocations. Figure 9 illustrates the duration (cycles) and the amount of transferred pixels and weights in the DMA interval. Figure 9 exemplifies two CSs, where each per-type bank allocation is named based on the bank map of the data types, one 3-tuple (B2, B1, B0) for each data type. For instance, 4O2W1I refers to the per-type bank allocation that allocates the output feature maps (O) to B2, the filters (W) to B1 and the input feature maps (I) to B0.

![Figure 9](image_url)

**FIGURE 9.** Examples of CSs: 4O2W1I with (a) three DMACs, (b) two DMACs.

Figure 9 (a) shows a CS with the three-master bus interface consisting of RI-DMAC, RW-DMAC, and WO-DMAC (referred to as 3M-4O2W1I). As the set of activated DMACs varies across DMA intervals within a processing pass, the CP also varies across DMA intervals. For example, as shown in Figure 9 (a), in the case of 3M-4O2W1I, the CP of the DMA interval 0 is 4W since the WO-DMAC is the only activated DMAC and is allocated B2. The duration of this DMA interval is determined by the DMA set time of RI-DMAC (assumed to be 80 cycles in this example). Since RI-DMAC has been initiated by the processor core, the CP of the DMA interval 1 is 4W1R since the RI-DMAC is the second activated DMAC and is allocated B0. Likewise, the duration of the DMA interval 1 is determined by the DMA set time of RW-DMAC (assumed to be 160 cycles in this example). Once RW-DMAC is initiated, it becomes the third activated DMAC, ending up with 4W2R1R since it is allocated B1. Finally, 3M-4O2W1I can be expressed as a set of CPs, in this case, 4W, 4W1R, 4W2R1R, 4W1R, and 4W.

Figure 9 (b) shows another example of a CS with the two-master bus interface consisting of R-DMAC and WO-DMAC (referred to as 2M-4O2W1I). A single read DMAC (R-DMAC) is assumed to transfer input feature map and filter. Therefore, the CP set of this CS may be different from that of 3M-4O2W1I and consists of 4W, 4W1R, 4W2R and 4W.

Note that there may be different CP sets of a given CS, depending on how a processing pass progresses, more accurately, how a processing pass is divided into multiple DMA intervals. For example, the CP set may vary with tile size since the communication amount of each DMAC depends on the tile size, as shown in Figure 10. In more detail, 3M-5O6W3I is expressed as 5W, 3R, 3R6R, and 3R in Figure 10 (a), whereas it is expressed as 5W, 5W3R, 5W3R6R, 5W6R, and 6R in Figure 10 (b). In fact, the CP set may vary with several other parameters, for example, the accelerator frequency that also affects how to divide a processing pass into DMA intervals.

![Figure 10](image_url)

**FIGURE 10.** Dependence of CP sets on tile size (3M-5O6W3I) when the communication is dominated by (a) input feature maps or (b) filters.

However, regardless of how a processing pass is divided into DMA intervals, for example, regardless of tile size, a finite set of CPs suffices to express a given CS. In other words, a CP set of a given CS is always given as one of the subsets. For example, 3M-5O6W3I can be expressed by a set of 7 CPs, i.e., 3R, 6R, 6R3R, 5W, 5W3R, 5W6R, and 5W6R3R. Here note that equivalent CPs are counted as a single CP since they always have the same communication bandwidth. Moreover, the CP sets of different CSs may have many constituent CPs in common. For example, 3R, 6R, 6R3R, 5W, 5W3R, 5W6R and 5W6R3R are common to both 3M-5O6W3I and 3M-6O5W3I. Therefore, it can...
be concluded that there is a finite set of CPs that suffices to express a CS. As a matter of fact, the CSs of interest in Figure 11 can be expressed as a subset of 31 CPs shown in Figure 12.

Before leaving this section, it should be mentioned that the definitions of CP and CS do not change for every application. As mentioned earlier, CP and CS are defined based on the number of DMACs and bank allocations, more importantly, independently of the application. For example, we have applied the same set of CSs (i.e., those present in Figure 11) to two different applications - CNNs and wireless communications (as will be shown in the later section). In other words, all the CPs and CSs considered in this paper can be generally applied to any applications. Given the accelerator-application pair, i.e., if both accelerator and application are specified, the number of DMACs is given by the accelerator and thus only a subset of CSs (i.e., a set of CSs with the given number of DMACs) can be considered. In addition, given an application, each CS is expressed by a set of CPs, one CP for each DMA interval and, more importantly, the communication bandwidth of each DMA interval is determined by the corresponding CP. In detail, the communication bandwidth often improves with the number of DMACs and the number of available banks, mainly thanks to more parallelism. However, it is the CS that determines the CP for each DMA interval. In other words, for a given CS, it is not any more possible to choose the optimum CP (i.e., the CP that maximizes the communication bandwidth) for each DMA interval. For the reason, we focus on the optimizations of CS (not the optimizations of CP) in this paper.

VII. PERFORMANCE ESTIMATION ALGORITHMS

In this paper, in order to estimate the performance of a CS for a DMA-controlled accelerator, the communication bandwidths of the constituent CPs are obtained using prior simulations. This enables us to estimate the communication bandwidth of CPs accurately by taking into account the impact of DRAM latency and bus protocol overhead, as mentioned in Section V. However, the main downside of simulation-based approach is that it often takes a prohibitively long time to simulate, as mentioned previously.

Even with full-system simulators of high abstraction-levels (e.g., transaction-level simulator proposed in [24]), it may be too cumbersome to explore a broad design space. For example, according to our experiments, it takes about a few hours to evaluate a single CS with hundreds of different combinations of tile sizes and accelerator frequencies.

In order to minimize the simulation time, simulation-based approach is constrained to the performance estimation of a set of CPs. Since a few tens of CPs suffice to express most of the CSs of interest as depicted in the previous section, the extra simulation time becomes negligible, particularly, in the case of a broad design space (i.e., a space of hundreds of design points). Once the communication bandwidths of CPs are obtained from prior simulations, it is possible to estimate the performance of any CS on a per DMA interval basis, importantly, without any additional simulations, since each CS can be expressed as a CP set. For example, in Figure 9 (b), the performance of 2M-4O2W1I can be simply estimated by the communication bandwidths of the four constituent CPs, 4W, 4W1R, 4W2R, and 4W. Figure 13 illustrates how to estimate the performance of the CS on a per DMA interval basis. In Figure 13 (a), in the DMA interval 0, WO-DMAC is the only activated DMAC and is allocated B2. This DMA interval ends by the completion of setting up R-DMAC and thus the duration is determined by the corresponding set time.
The communication amount of WO-DMAC within this DMA interval is calculated using the communication bandwidth of the corresponding CP (i.e., 4W). Note that WO-DMAC remains to be activated in the next DMA interval since the remaining communication amount is non-zero. In the DMA interval 1, R-DMAC is also activated DMAC since it has already been set up. As shown in Figure 13 (b), R-DMAC and WO-DMAC are allocated B2 and B0, respectively. This DMA interval continues until the transfer of input feature maps by R-DMAC is completed (i.e., the remaining communication amount becomes zero). Therefore, the duration of the DMA interval 1 can be calculated by using the communication bandwidth of the corresponding CP (i.e., 4W1R) and the remaining communication amount of input feature maps. In the next DMA interval, R-DMAC starts the transfer of filters. Figure 13 (c) shows that R-DMAC and WO-DMAC are still activated DMAC and are allocated B2 and B1, respectively. This DMA interval continues until the transfer of filters by R-DMAC is completed and R-DMAC is deactivated. This is similar to the schedule-aware performance estimation in [23] in the sense that it keeps track of the progress of a processing pass (e.g., the remaining communication amount per DMAC) on a per DMA interval basis. However, instead of relying on the static analysis in [23], the proposed primitive-based performance estimation algorithm obtains the communication bandwidths of a set of CPs from full-system simulations. This makes it easier to take into account the impact of several design parameters (i.e., bus burst length, accelerator/DRAM frequency etc.).

Algorithm 1 presents the details of the primitive-based performance estimation algorithm. The optimizations of CS can be generally applied to any number of DMACs (e.g., 1 or more than 3) although the number of DMACs is assumed to be 2 or 3 in this section. Moreover, the proposed performance estimation algorithm (Algorithm 1) can be readily generalized to any number of DMACs. For the sake of brevity, the use of three DMACs is assumed in Algorithm 1. Given the inputs of Algorithm 1, namely, the number of pixels and weights (NoPW), the per-type bank allocation (BA), the DMA set time (Tset) and the scaling factor (scale), the algorithm first updates the remaining communication amount (NoPW) of the present DMA interval through the communication amount of the DMA interval (dNoPW) (line 2). Subsequently, it checks whether each DMAC has been set (setDma) or activated (actDma) (line 3). If there remain pixels or weights (lines 5-7), it goes to the next DMA interval (i.e., the i-th DMA interval) (line 8). For the next step, it takes the bank allocation accessed by activated DMACs (actBA) (line 9). For example, in Figure 9 (a), it takes the bank allocation of 4W and 4W1R in the DMA interval 0 and 1, respectively. Subsequently, the communication bandwidth needs to be scaled by the
weights (NoPW), the per-type bank allocation (BA), the DMA set time (Tset) and the scaling factor (scale), the algorithm first updates the remaining communication amount (NoPW) of the present DMA interval through the communication amount of the DMA interval (dNoPW) (line 2). Subsequently, it checks whether each DMAC has been set (setDma) or activated (actDma) (line 3). If there remain pixels or weights (lines 5-7), it goes to the next DMA interval (i.e., the i-th DMA interval) (line 8). For the next step, it takes the bank allocation accessed by activated DMACs (actBA) (line 9). For example, in Figure 9 (a), it takes the bank allocation of 4W and 4W1R in the DMA interval 0 and 1, respectively. Subsequently, the communication bandwidth corresponding to the CP is returned from the predefined function (BA2BW), which is responsible for searching the predefined set of CPs (e.g., 31 CPs in Figure 10) for one of the equivalent CP (line 10). For example, in the DMA interval 0 of Figure 7 (a), it returns the communication bandwidth of 1W (which is equivalent to 4W). In the presence of partial bursts, the communication bandwidth needs to be scaled by the
burst efficiency. For example, for the maximum bus burst length of 8 and the bus burst length of 15, the communication bandwidth should be scaled by the burst efficiency of 15/16. Next, based on the scaled communication bandwidth of the CP (commBW), it calculates the cycles of the DMA interval (T). In more details, when the amount of pixels or weights to be transferred in the processing pass is divided using the scaled bandwidth, the time taken to transfer each data type is calculated (Tptdt) (line 11). Then, the duration of the DMA interval is determined to be smaller between the transfer time for each data type (timT) and the DMA set time (Tset) (lines 12-13). Given the duration of the DMA interval (T), it updates the set time remaining in each DMAC (Tset) (line 14) and calculates the communication amount of the DMA interval (dNoPW) based on the communication bandwidth of the CP (line 15).

In summary, the duration of each DMA interval is determined by the remaining communication amount of each DMAC and the communication bandwidth of the DMA interval if the transfer time is smaller than the DMA set time. The proposed performance estimation algorithm keeps track of the remaining communication amounts on a per DMA interval basis, i.e., by updating the communication amounts of each DMA interval. In addition, based on the set of activated DMACs of each DMA interval, the proposed algorithm determines the CP of each DMA interval, which in turn determines the communication bandwidth of the DMA interval.

VIII. PERFORMANCE EVALUATION

This section evaluates the communication performance of a DMA-controlled accelerator. The proposed performance estimation algorithm is verified against the full-system simulator [24]. Each hardware block of the system depicted in Figure 1 (a) is modelled using SystemC-TLM, except the DRAM subsystem that is modelled using DRAMSim2, an open-source DRAM simulator [39]. As is typical in the SystemC-TLM simulators (e.g., [15], [46]), the full-system simulator provides the capability to skip the actual processing inside an individual module for simulation speedup (e.g., computation of the MAC array) while maintaining the cycle-accurate timing on the module boundary, i.e., on the socket level. The sockets make use of the TLM2.0 core interfaces to communicate between hardware blocks. In general, the approximately-timed (AT) coding style based on non-blocking transports is chosen for data transfer whereas the loosely-timed (LT) coding style based on blocking transports is chosen for control transfer. This makes it possible to model every transfer between hardware blocks in an event-driven yet cycle-accurate manner. For example, it is possible to keep track of each transfer of the bus request between DMACs and DRAM subsystem in a cycle-accurate manner. In order to model the AXI protocol efficiently, the GSGP sockets [47] as well as the TLM2.0 sockets are used. More specifically, the GSGP sockets are used for AXI4-MM while the TLM2.0 sockets are for AXI-Lite. In the case of AXI4-MM, the six phases of the GSGP generic protocol are mapped into the corresponding handshake signals (i.e., valid and ready) of the five AXI channels AR, AW, R, W and B. Note that such a low-level modeling enables us to evaluate the impact of several protocol-related parameters such as the number of outstanding transactions. The accelerator includes a SystemC thread that exchanges a set of control signals with the submodules such as DMACs, MAC array and on-chip SRAM, for example, with the stream interface for double buffering. The corresponding control transfer is implemented using the LT coding style based on blocking transport, as mentioned before.

The communication bandwidth of the measured CPs is provided to the primitive-based estimation algorithm, and the performance of the measured CSs is compared to the estimated performance through the primitive-based estimation algorithm. The design space assumed in this section is summarized in Table 1. When it comes to the DRAM controller, the scheduling is assumed to be the first ready, first come first serve (FR-FCFS) with open-page mode together with the 4-time close [48]. In addition, the address mapping policy of column-row-bank-rank-channel is assumed. The hardware accelerator is assumed to operate at 2× clock frequency (e.g., 1.06 GHz), compared to the off-chip DRAM (e.g., 533 MHz), as assumed in [30]. As mentioned earlier, regarding the data layout, within a bank, the pixels of a feature map are assigned to the DRAM locations in a row-major order. Although the proposed primitive-based estimation algorithm is generally applicable to any DMA-controlled accelerators, the third convolutional layer of AlexNet [38] and the fifth convolutional layer of MobileNet [49] are assumed as the example applications in this section. For the convolutional layers, the loop tiling is set according to the three different tile sizes given in the Table 2 (named tAlex1, tAlex2 and tMobile) such that the hardware accelerator is limited by the communication bandwidth, i.e., being communication-limited. In order to prove that the proposed performance estimation algorithm is generally applicable to any other application, we have included additional experimental results on CNNs (ResNet-50 [50] and 3D U-Net [51]) and wireless communications (LDPC-coded MIMO-OFDM [52], [53]). In the case of CNNs, the 13 layers for ResNet-50 and the 18 layers for 3D U-Net are assumed as the example applications in this section. In addition, in the case of wireless communications, the simulation conditions such as DRAM/bus parameters are set to be equal to those in Table 1. In addition, the tile sizes are replaced by the number of spatial streams (NSS) and the modulation coding scheme (MCS) given in Table 3.

| DRAM parameters | Value | Bus parameters | Value |
|-----------------|-------|----------------|-------|
| Memory interface | FR-FCFS | Memory interface | FR-FCFS |
| Page mode | Open page | Max. burst size | 2 |
| Address mapping | Bank-Row-Col | No. of outstanding transactions | 6 |
| Max. burst length | 8 | Outstanding transactions latency | 1 |
| [CL, ROCD, IRP] | [7, 7, 7] | | |
TABLE 2. Layer shapes and tile sizes of the convolutional layers considered in this paper.

| Layer shape parameters | Tile parameters |
|------------------------|-----------------|
| M | C | I/F | E/F | R | S | TM | TC | TE | TF |
| tAlex1 | 384 | 256 | 15 | 3 | 3 | 3 | 64 | 2 | 13 | 13  
| tAlex2 | 384 | 256 | 15 | 3 | 3 | 3 | 24 | 32 | 13 | 13  
| tMobile | 128 | 64 | 36 | 36 | 1 | 1 | 64 | 16 | 16 | 16  

TABLE 3. Communication amount required for FFT, MIMO and LDPC to process an OFDM symbol.

| MCS 4 | FFT | MIMO | LDPC |
|-------|-----|------|------|
| R-DMAC | W-DMAC | R0-DMAC | R1-DMAC | W-DMAC | R0-DMAC | R1-DMAC | W-DMAC |
| NS8 1 | 1024 | 1024 | 1920 | 1920 | 1920 | 1920 | 960 | 960 | 183 |
| NS8 2 | 2048 | 2048 | 1920 | 1920 | 1920 | 1920 | 960 | 960 | 183 |
| NS8 4 | 4096 | 4096 | 1920 | 1920 | 1920 | 1920 | 960 | 960 | 183 |

A. EVALUATION OF BANK INTERLEAVING

Figure 14 shows the impact of DRAM access pattern, i.e., bank interleaving on the communication bandwidth. The figure shows that the optimum bank interleaving improves the communication bandwidth significantly, for example, in Figure 14, by 39.6% with up to six outstanding transactions. In addition, it is clearly shown that the bank interleaving set to the number of outstanding transactions guarantees almost the optimum bandwidth. For example, in the case of two outstanding transactions, the optimum bank interleaving is shown to be two, regardless of the accelerator operating frequency. This is not surprising at all since the optimum bank interleaving is often related to the number of bus requests that can be executed with a single page open - the number of outstanding transactions. An exception is the case with six outstanding transactions: the optimum bank interleaving is four, i.e., smaller than the number of outstanding transactions. This is due to the assumption of the 4-time close policy: a single page open can accommodate no more than 4 additional reads or writes [48]. For this reason, we set the bank interleaving to the number of outstanding transactions in the remainder of this section.

B. EVALUATION OF OUTSTANDING TRANSACTIONS

Figure 15 shows that the communication bandwidth generally increases with the number of outstanding transactions. For example, in Figure 15, the doubled number of outstanding transactions (e.g., increased from two to four) improves the communication bandwidth by 2×. The figure also shows that, in the communication-limited case, the communication bandwidth tends to be proportional to the number of outstanding transactions. However, the use of six outstanding transactions does not provide the same bandwidth gain since the communication bandwidth is limited by DRAM latency (rather than limited by bus protocol overhead). Taking into account the trade-off between bandwidth gains and hardware overhead, a reasonable design choice may be to support four outstanding transactions. In addition, the figure shows that, for the given number of bus masters, the bank allocation may provide an additional bandwidth gain. For example, in Figure 15, assuming four outstanding transactions and two bus masters, the communication bandwidth varies from 0.6 to 1.0 depending on the bank allocation. More specifically, 1R2R outperforms 3R3R and 3R2R thanks to the reduction of bank collision.

C. EVALUATION OF COMMUNICATION PRIMITIVES

Figure 16 shows the communication bandwidths of a few CPs. As explained earlier, the proposed primitive-based performance estimation obtains them by the full-system simulator in [24]. As previously depicted in Figure 2, an extra DMAC generally improves the communication bandwidth. For example, 1R2R providing a 64% gain over 1R. It is also shown that, given the number of DMACs, the bank allocation may improve the communication bandwidth further. For instance, in the case of 1R6R, the bandwidth gain due to the bank allocation amounts to 68% over that of 1R1R. In detail, 1R6R is shown to be the best per-type bank allocation: it has no bank collision between the R0-DMAC and R1-DMAC, and furthermore it assigns two banks to the R1-DMAC.
Figure 16 also compares the proposed primitive-based performance estimation (i.e., full-system simulation) with the conventional analysis-based estimation mentioned in Section V, the CAG-based bus performance estimation in [16] and the statistics-based DRAM latency estimation in [17]. In the CAG-based bus performance estimation, the communication bandwidth of each DMAC is inversely proportional to the number of (activated) DMACs, assuming equal priority arbitration. As pointed out in Section V, this can predict the communication bandwidth accurately in case of bus-limited accelerator. Figure 16 also shows that it matches well with the simulation results when the (actual) communication bandwidth is close to unity (i.e., in the bus-limited case), e.g., in the case of 7R7R, 1R6R, and 1R2R. However, it results in a serious estimation error when the communication bandwidth is not sufficiently high and thus is limited by DRAM latency, e.g., in the case of 1R1R and 1W1R1R. Besides, the statistics-based performance estimation estimates the DRAM latency based on the statistics of different access conditions (e.g., row buffer hit). However, as mentioned in Section V, it is not straightforward to apply it to the case with multiple DMACs and multiple DRAM banks. For the purpose of comparison, this paper extends the statistics-based performance estimation in [17] by assuming that the communication bandwidth is equally split among DMACs [16]. For example, the figure shows that, in the case of 1R1R, the communication bandwidth of 1R is equally split among two DMACs. However, this is not always the case, for example, when there are multiple DMACs or multiple DRAM banks, e.g., in the case of 7R7R. This justifies the accuracy of the simulation-based approach taken by the proposed primitive-based performance estimation algorithm. Thus it can be concluded that the proposed estimation always outperforms the conventional estimation in terms of estimation accuracy since it causes no performance estimation error. Therefore, the communication bandwidth difference between the proposed estimation and the conventional ones should be seen as the estimation error of the conventional estimation. In fact, the figure shows that the conventional estimation in [16] and [17] tends to overestimate the communication bandwidth. In other words, the estimated communication bandwidth of the conventional estimation is often larger than the actual communication bandwidth (that of the proposed estimation), as shown in Figure 16. This can be explained by the fact that the conventional estimation in [16] and [17] takes into account either DRAM latency or bus protocol overhead, but not both.

Assuming the third convolutional layer of AlexNet (tAlex2), the performances of a couple of CSs are evaluated in terms of the duration of a processing pass in Figure 17. The communication bandwidths of the constituent CPs are taken from the proposed primitive-based performance estimation, the CAG-based bus performance estimation [16] and the statistics-based DRAM latency estimation [17]. For the sake of comparison, the figure also includes the conventional simulation-based approach that runs a full-system simulation, once for each of the CSs. It is shown that the proposed estimation matches quite well with the simulation-based approach (i.e., full-system simulator in [24]) with an error of less than 5%. In contrast, the conventional estimation in [16] and [17] shows a non-negligible estimation error compared to the simulation-based approach due to the erroneous estimation of communication bandwidth shown in Figure 16. For example, in the case of 1O1W1I, the conventional estimation in [16] overestimates the communication bandwidth by 60% since it does not take into account DRAM latency due to bank collision. In addition, in the case of 4O2W1I, the conventional estimation in [17] overestimates the communication bandwidth by 21%. This can be explained by the fact that the conventional estimation in [17] does not take into account bus

![Figure 16](image-url)

**FIGURE 16.** Communication bandwidth of CPs obtained by CAG-based [16], statistics-based [17], simulation-based [24], and (proposed) primitive-based estimation.

![Figure 17](image-url)

**FIGURE 17.** Communication performance of CSs based on CAG-based [16], statistics-based [17], simulation-based [24], and (proposed) primitive-based estimation.
E. EVALUATION OF COMMUNICATION SCHEMES

Figure 19 compares the performance of several CSs in terms of the duration of a processing pass. Above all, the overall communication performance improves with the optimized bank allocation. For example, in Figure 19 (c), the comparison between the CS with three DMACs, e.g., 3M-1O1W1I and 3M-7O7W7I, shows that the optimized bank allocation improves the communication performance by up to 68%. Moreover, it is shown that the optimality of CS generally depends on both the layer shape and the tile size of the convolutional layer. For example, 2M-7O7W7I provides the near-optimal performance in the case of tMobile whereas it does poor performance in the case of tAlex1 and tAlex2. In addition, compared with the full-system simulations, the proposed primitive-based performance estimation can predict the performance of CSs with an error no more than 5%. The estimation error turns out to be caused mainly by extremely short DMA intervals since the proposed performance estimation algorithm assumes a sufficiently long DMA interval for each CP. In the remainder of this section, some of the interesting observations made in Figure 19 are explained.

First, it should be noted that the use of an extra DMAC does not necessarily improves (and may even degrade) the performance unless the bank allocation is carefully chosen. In other words, the communication performance with three DMACs may not be larger than that with two DMACs. For example, in Figure 19 (a), the processing pass of 2M-6O6W3I is 6% shorter than that of 3M-4O2W3I. This can be explained by additional bank-level parallelism of 2M-6O6W3I (particularly in the last DMA interval), as shown in Figure 20 (a).

Another interesting observation is that the optimality of bank allocation may depend on the number of the available DMACs. For example, in Figure 19 (b), 3M-4O2W1I generally requires less communication time than other CSs with three DMACs, whereas 2M-4O2W1I generally requires more communication time than other CSs with two DMACs. This can be understood by recalling that, as shown in Figure 20 (b), the absence of bank-level parallelism of 4O2W1I can be compensated by allowing multiple DMACs to run simultaneously. For example, Figure 20 (b) shows that the use of two read DMACs allows 3M-4O2W1I to exploit bank-level parallelism most of the time (i.e., except in the first and the last DMA intervals).

Furthermore, for the same per-type bank allocation, the three-DMAC bandwidth may be smaller than the two-DMAC bandwidth. For example, in Figure 19 (c), 4O2W1I requires more communication time with three DMACs than with two DMACs. This can be largely explained by the fact that the W-DMAC experiences a smaller communication bandwidth in DMA interval 2 in Figure 9 (a) (4W2R1R) is lower than in DMA interval 2 in Figure 9 (b) (4W2R).

Figure 19 includes the performance limit of the CSs. Instead of the per-type bank allocation assumed in Section VI, the CP of each DMA interval is chosen to maximize the communication bandwidth for the given set of activated DMACs. For example, in Figure 9 (a), the WO-DMAC is activated in DMA interval 0 and thus 7W is chosen as the CP and, likewise, in DMA interval 1, since the RI-DMAC has been activated additionally, 1W6R is chosen to maximize the communication bandwidth. Figure 19 shows that most of the performance limit can be achieved by simply optimizing the CS using the proposed performance estimation algorithm.

Figure 21 shows the optimizations of CSs for the convolutional layers of two different neural networks, ResNet-50 [50] and 3D U-Net [51] assuming the set of 15 CSs in Figure 11. The figure includes the optimum CS of each convolutional layer together with the corresponding performance gain over CS0 (1O1W1I). It is shown that the optimum CS tends to
widely vary with layer shape and tile size. For example, in the case of ResNet-50, as shown in Figure 21 (a), the optimum CS of the first convolutional layer is CS11 (7O7W7I), whereas that of the third convolutional layer is CS14 (4O2W5I). Moreover, it is shown that the optimizations of CS provide the maximum performance gain of 67% and 66% for ResNet-50 and 3D U-Net, respectively.

Before leaving this sub-section, it should be mentioned that the optimizations of CP and CS change for every application. In other words, a CS is optimum for one application whereas it is not optimum for another application. Such application-dependent optimality of CS is verified by some of the simulation results provided in this paper. For example, 3M-4O2W5I is optimum for AlexNet (Figure 19 (b)), whereas it is not optimum for MobileNet (Figure 19(c)). One of the reasons for the application-dependent optimality is that the per-type communication amount may vary with the application. In the case of CNNs, the communication amount depends on not only the layer shape but also the tile sizes. In the aforementioned example, in Figure 19 (b) (tAlex2), the communication amount of I, W, and O is given by 7200, 6912 and 514, respectively, whereas, in Figure 19 (c) (tMobile), the communication amount of I, W, and O is given by 4096, 1024 and 8192, respectively. The difference of communication amount explains the relatively larger performance gain of allocating input feature maps multiple banks in Figure 19 (b) (tAlex2).

F. EXTENSION TO CONSECUTIVE LAYERS

In this sub-section, the proposed primitive-based performance estimation algorithm is extended to multiple consecutive convolutional layers of a CNN. Recalling that the output feature maps of a convolutional layer is nothing but the input feature maps of the next convolutional layer, the bank allocation of the output feature maps of a convolutional layer...
simply determines that of the input feature maps of the next convolutional layer. This inter-layer dependence should be considered in optimizing the CSs of the consecutive convolutional layers. If multiple consecutive convolutional layers are jointly optimized, for example, in terms of tile size, the corresponding design space may be too broad to explore using a full-system simulator (e.g., [24]). In this case, the proposed primitive-based performance estimation algorithm can facilitate the exploration of the design space significantly.

**FIGURE 22.** Communication performance of five convolutional layers of AlexNet with respect to CS.

Figure 22 shows the communication performance of the five convolutional layers of AlexNet. Given a pre-defined tile size, a set of 15 CSs in Figure 11 is considered for each convolutional layer. In the figure, all the possible combinations of CSs that satisfy the inter-layer dependence are sorted in terms of communication performance. It is shown that the optimum combination of CSs improves the communication performance by up to 34%. In more detail, the optimum combination of CSs is given as 4O2W1I, 4O6W1I, 6O3W1I, 4O2W1I, and 1O2W1I (from the first convolutional layer). This implies that the optimality of CS generally depends on the both the layer shape and the tile size of the convolutional layer, as shown in Figure 19.

**G. EXTENSION TO OTHER APPLICATIONS**

In this subsection, the optimizations of CSs through the proposed primitive-based performance estimation algorithm are extended to wireless communications (LDPC-coded MIMO-OFDM [52], [53]) applications. As shown in Figure 23, the LDPC-coded MIMO-OFDM consists of five hardware blocks, including initial synchronization, fast Fourier transform (FFT), channel estimation, multiple-input and multiple-output (MIMO) and low-density parity-check code (LDPC). The communication amount for each application depends on the NSS and the MCS given in Table 3.

As shown in Figure 24, FFT, MIMO, and LDPC are implemented as a DMA-controlled hardware accelerators. In more detail, Figure 24 (a) depicts an FFT accelerator comprised of two DMACs, each of which is dedicated to one of the two data types of the FFT accelerator. A read DMAC for I/Q-phase (R-DMAC) and a write DMAC for I/Q-phase (W-DMAC) are used. Figure 24 (b) shows a MIMO accelerator consisting of three DMACs. A read DMAC for I/Q-phase from FFT accelerator (R0-DMAC), a read DMAC for I/Q-phase from channel estimator (R1-DMAC) and a write DMAC for codeword to LDPC (W-DMAC) are used. Lastly, Figure 24 (c) describes an LDPC accelerator consisting of...
three DMACs, where each of the DMACs is dedicated for one of the three data types of the LDPC accelerator. A read DMAC for odd codeword (R0-DMAC), a read DMAC for even codeword (R1-DMAC) and a write DMAC for coded codeword (W-DMAC) are used.

Figure 25 shows the performance estimation of optimizations of CSs for each application of LDPC-coded MIMO-OFDM in MCS 4 and NSS 2, such as FFT, MIMO and LDPC assuming the set of 15 CSs in Figure 11. The figure includes the optimum CS of each application together with the corresponding performance gain over CS0. It is shown that the optimum CS tends to widely vary with the communication amount of each DMAC of hardware accelerators. For instance, as shown in Figure 25, in the case of the LDPC accelerator, the optimum CS is 14. However, in the case of FFT and MIMO accelerator, the optimum CSs are 13 and 8, respectively. This can be explained by the fact that the assigned communication amount of W-DMAC is dominant compared to that of R0-DMAC and R1-DMAC. It implies that the parallelism of banks allocated to W-DMAC facilitates improving the performance of the LDPC accelerator. In addition, the simulation results show that the optimizations of CS provide the maximum performance gain of 56%, 60% and 57% for FFT, MIMO and LDPC, respectively.

IX. CONCLUSION

In this paper, the performance of CSs characterized by the number of DMACs and the bank allocation of DRAM is evaluated and optimized. The newly proposed estimation algorithm of this paper can estimate the communication performance of a CS based on the communication bandwidths of the constituent CPs of the DMA intervals. When applied to the design space exploration of CNNs and wireless communications, it is shown that the proposed primitive-based performance estimation algorithm achieves an estimation error of less than 5% and 4%, respectively. Compared with the conventional simulation-based approach, the proposed performance estimation algorithm provides a simulation speedup of two orders of magnitudes. In addition, the bank interleaving is proposed to be set to the number of outstanding transactions. It is also shown that the optimized CS can improve communication performance significantly, for example, by up to 68% for the third convolutional layer of AlexNet and 60% for the MIMO of LDPC-coded MIMO-OFDM. Moreover, a couple of interesting observations are made as to how to optimize the CS of DMA-controlled accelerators. For example, the simulation results show that the optimum CS is application-dependent. It is also shown that the use of an extra DMAC does not necessarily improve the communication performance.

Lastly, it is worthwhile to mention that the proposed estimation algorithm is generally applicable to any communication-limited DMA-controlled accelerators. In particular, the extension of the proposed CS optimization with performance estimation algorithm into the emerging compute-in-memory (CiM) hardware accelerator is considered to be promising for future work. Note that such a CiM-based hardware accelerator is often equipped with one or more DMACs [54]–[58]. In addition, as a standalone IP, it is connected to an off-chip memory through an on-chip bus [59], [60]. Moreover, the bank allocation of DRAM tends to affect the communication performance of the emerging CiM-based hardware accelerator, as depicted in [61], [62]. Thus, we expect the performance estimation algorithm proposed in this paper to be generally applicable to the CiM-based hardware accelerator.

REFERENCES

[1] B. Khabbazan and S. Mirzakuchaki, “Design and implementation of a low-power, embedded CNN accelerator on a low-end FPGA,” in Proc. 22nd Euromicro Conf. Digit. Syst. Design (DSD), Aug. 2019, pp. 647–650.

[2] F. Spagnolo, S. Perri, F. Frustaci, and P. Corsonello, “Designing fast convolutional engines for deep learning applications,” in Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS), Dec. 2018, pp. 753–756.
[44] T. Marty, T. Yuki, and S. Derrien, “Safe overclocking for CNN accelerators through algorithm-level error detection,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 39, no. 12, pp. 4777–4790, Dec. 2020.

[45] (Sep. 2018). MT41K128M16JT-125 Device Data Sheet. [Online]. Available: https://www.micron.com/media/client/global/documents/products/memory/mt41k128m16jt-135v_ddr3/pdf

[46] Imperas. Open Virtual Platforms (OVP). Accessed: 2012. [Online]. Available: https://www.ovpworld.org

[47] GreenSoCs. GreenSoCs Virtual Platforms. Accessed: 2018. [Online]. Available: https://www.greensocs.com/

[48] Y.-C. Lee, T.-S. Hsu, C.-T. Chen, J.-J. Liou, and J.-M. Lu, “NNSim: A fast and accurate SystemC/TLM simulator for deep convolutional neural network accelerators,” in *Proc. Int. Symp. VLSI Design, Autom. Test (VLSI-DAT)*, Apr. 2019, pp. 1–4.

[49] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” 2017, arXiv:1704.04861, [Online]. Available: http://arxiv.org/abs/1704.04861

[50] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015, arXiv:1512.03385, [Online]. Available: https://arxiv.org/abs/1512.03385

[51] O. Ciçek, A. Abdulkadir, S. S. Lienkamp, T. Brox, and O. Ronneberger, “3D U-Net: Learning dense volumetric segmentation from sparse annotation,” in *Proc. Int. Conf. Med. Image Comput. Comput.-Assist. Intervent. (MICCAI)*, Oct. 2016, pp. 424–432.

[52] P.-Y. Tsai, P.-C. Lo, F.-J. Shih, W.-J. Jau, M.-Y. Huang, and Z.-Y. Huang, “A 4 x 4 MIMO-OFDM baseband receiver with 160 MHz bandwidth for indoor gigabit wireless communications,” *IEEE Trans. Circuits Syst. I, Reg. Papers, Reg. Papers*, vol. 62, no. 12, pp. 2929–2939, Dec. 2015.

[53] T. Suzuki, H. Yamada, T. Yamagishi, D. Takeda, K. Horisaki, T. Vander Aa, T. Fujisawa, L. Perre, and Y. Unekawa, “High-throughput, low-power soft-definition radio using reconfigurable processors,” *IEEE Micro*, vol. 31, no. 6, pp. 19–28, Nov. 2011.

[54] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, “A programmable embedded microprocessor for bit-scalable in-memory computing,” *IEEE J. Solid-State Circuits*, vol. 55, no. 9, pp. 2609–2621, Sep. 2020.

[55] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, “XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks,” *IEEE J. Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, Jun. 2020.

[56] W.-S. Khwa, J.-J. Chen, J.-F. Li, X. Si, E.-Y. Yang, X. Sun, R. Liu, P.-Y. Chen, Q. Li, S. Yu, and M.-F. Chang, “A 65 nm 4 Kb algorithm-dependent computing-in-memory SRAM unit-macro with 2.3 ns and 55.8 TOPS/W fully parallel product-sum operation for binary DNN edge processor,” in *IEEE Int. Solid-State Circuits Conf. (ISSCC)* Dig. Tech. Papers, Feb. 2018, pp. 496–498.

[57] M. Zhu, Y. Zhuo, C. Wang, W. Chen, and Y. Xie, “Performance evaluation and optimization of HBM-enabled GPU for data-intensive applications,” in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1245–1248.

[58] Z. Wang, H. Huang, J. Zhang, and G. Alonso, “Shuhai: Benchmarking high bandwidth memory on FPGAs,” in *Proc. IEEE 28th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, May 2020, pp. 111–119.

[59] G. Singh, D. Diamantopoulos, C. Haglétner, J. Gomez-Luna, S. Stuijk, O. Mutlu, and H. Corporaal, “NERO: A near high-bandwidth memory on FPGAs,” in *Proc. Int. Symp. Field-Programmable Logic Appl. (FPL)*, Aug. 2020, pp. 9–17.

[60] A. Kurth, W. Rönninger, T. Benz, M. Cavalcante, F. Schuiki, F. Zaruba, and L. Benini, “An open-source platform for high-performance non-coherent on-chip communication,” 2020, arXiv:2009.05324, [Online]. Available: http://arxiv.org/abs/2009.05324

[61] P. Gu, X. Xie, S. Li, D. Niu, H. Zheng, K. T. Malladi, and Y. Xie, “DLUX: A LUT-based near-bank accelerator for data center deep learning training workloads,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 40, no. 8, pp. 1586–1599, Aug. 2021.

[62] E. Azarkhish, D. Rossi, L. Loi, and L. Benini, “Neurostream: Scalable and energy efficient deep learning with smart memory cubes,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 2, pp. 420–434, Feb. 2018.

**JOOHO WANG** received the B.S. degree in electronics engineering from Korea Polytechnic University (KPU), Siheung, South Korea, in 2014. He is currently pursuing the M.S./Ph.D. degree in electronics engineering with Konkuk University, Seoul, South Korea. His research interests include hardware/software co-design of programmable accelerators and simulation for SoC architecture.

**SUNGYOUNG PARK** (Senior Member, IEEE) received the Ph.D. degree in electronics engineering from Seoul National University, South Korea, in 2002. From 2002 to 2006, 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 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.