Data Transfer Modeling and Optimization in Reconfigurable Multi-Accelerator Systems

Alberto Ortiz, Alfonso Rodríguez, Andrés Otero and Eduardo de la Torre
Centro de Electrónica Industrial
Universidad Politécnica de Madrid
Madrid, Spain
{alberto.ortiz, alfonso.rodriguezm, joseandres.otero, eduardo.delatorre}@upm.es

***Abstract*—The use of accelerator-centric processing architectures in different application scenarios, ranging from the cloud to the edge, is nowadays a reality. However, the always increasing stringent operating conditions and requirements continues to push the research around hardware-based processing architectures, which are able to provide medium to high computing performance capabilities while at the same time supporting energy-efficient execution. In addition, reconfigurable devices (i.e., FPGAs) provide another degree of freedom by enabling software-like flexibility by time-multiplexing the computing resources. Nevertheless, bus-based computing platforms still face architectural bottlenecks when data transfers are not handled efficiently. In this paper, the communication overhead in a reconfigurable multi-accelerator architecture for high-performance embedded computing is analyzed and modeled. The obtained models are then used to predict the acceleration perfomance and to evaluate two different patterns for data transfers: on the one hand, a basic approach in which data preparation and DMA transfers are executed sequentially; on the other hand, a pipelined approach in which data preparation and DMA transfers are executed in parallel. The evaluation method is based on well-known accelerator benchmarks from the MachSuite suite. Experimental results show that using a pipelined data management approach increases performance up to 2.6x when compared to the sequential alternative, and up to 26.46x when compared with a bare-metal execution of the accelerators (i.e., without using the reconfigurable multi-accelerator processing architecture nor an Operating System).**

***Index Terms*—FPGAs, Communication Modeling, Dynamic and Partial Reconfiguration, Hardware Architectures.**

I. Introduction

The use of hardware accelerators to speed up complex computations with high energy efficiency has been around for several years [1]. In this scenario, reconfigurable computing enables the use of application-specific logic that can be switched in and out of the Field Programmable Gate Array (FPGA) substrate, enabling levels of flexibility for hardware designs which are comparable to software alternatives, while maintaining the high performance inherent to dedicated circuits.

However, including highly optimized accelerators in the system is usually not enough to attain the expected level of performance, especially in schemes where accelerators are used to offload massively parallel routines or loops (the so called *hot spots*) while the rest of the application runs sequentially in a host microprocessor. In fact, complex on-chip communication infrastructures (e.g., Networks on Chip [2]) have been developed to mitigate the sometimes excessive overheads derived from data transfers between hardware and software components. In the case of bus-based systems, data serialization in the bus is usually the main bottleneck. At this regard, specific strategies need to be implemented to fully exploit the available bandwidth. An important fact to bear in mind when optimizing data transference from the host to the accelerators is the inherent randomness introduced by the Operating Systems, such as the Linux kernel, which can not guarantee execution times. Under these circumstances, modeling the on-chip data transactions is the first necessary step to optimize the acceleration infrastructure.

In this paper, a reconfigurable multi-accelerator processing architecture has been analyzed to develop a lightweight model, which quantifies the communication overheads already present in the processing architecture. It is kept simple to enable its usage on-chip, even at run-time. The model has been obtained and validated experimentally to deal with the non-determinism introduced by the Operating System. Based on the results provided by the model, an optimized pattern for DMA-based (i.e., Direct Memory Access) data transfers in the multi-accelerator processing architecture is also presented. The proposed strategy pipelines buffer management (i.e., data copies between user-defined memory and a DMA-enabled memory) and actual burst transfers using a DMA engine to move data in a hierarchical memory structure (i.e., global/external memory and local/internal memory in each hardware accelerator). Both the original data delivery approach and the proposed one have been tested using well-known accelerator benchmarks from the *MachSuite* suite, which provides different computing workloads with several data access patterns in a series of HLS-based (i.e., High Level Synthesis)) high-level accelerator descriptions.

In summary, the main contributions of this paper are the following:

- A benchmark-based evaluation of a reconfigurable multi-accelerator processing architecture.
- A model to predict the communication overhead in DMA-based data transfers.

This work was supported by the Spanish Ministry of Economy and Competitiveness under the project PLATINO, with reference number TEC2017-86722-C4-2-R

- A pipelined data management scheme that reduces the communication overheads by parallelizing data buffering and DMA burst transfers.

The rest of this paper is organized as follows. Section II provides an overview of the most commonly used benchmark suites for HLS-oriented hardware acceleration, together with a brief analysis of the current works in on-chip communication strategies. Section III presents the data-path models for ARTICo^3, the reconfigurable multi-accelerator processing architecture which serves as the baseline in this work. An optimized data transfer strategy that builds on top of the developed models is presented in Section IV, and Section V evaluates both the models and the data transfer optimizations using an HLS-oriented benchmark suite. Section VI summarizes the main conclusions of the work and highlights the future lines of work.

II. Related Work

First, a discussion on the different benchmark suites available for the evaluation of HLS-based hardware architectures is provided in this section. Then, it analyzes the main works existing in the state-of-the-art dealing with optimization strategies for data communication in HW/SW co-processing architectures.

A. Benchmark suites

The increasing tendency of developing hardware accelerators through HLS-based methods and tools makes for the need of fair and re-usable performance comparisons between solutions. Benchmark suites, which are well known in the software and computer architecture domains, appear to cope with these requirements. However, there are not many different options available in the literature when talking about HLS. Among the different possibilities, three main options with a more extended or promising use, have been identified: CHStone [3], MatchSuite [4] and Rosetta [5].

Every benchmark suite is composed of a set of HLS kernels with different resource characteristics in order to evaluate all possible performance bottlenecks in the execution. The difference between the benchmark suites lies mostly in the amount of kernels available per suite and their actual complexity. This last difference is important when implementing the benchmarks in an accelerator-centric architecture such as ARTICo^3, as the kernels need to be complex enough to simulate real-life applications, while fitting in the limited amount of resources available in the architecture.

In this paper, the *MatchSuite* suite has been selected for testing purposes due to being the best when evaluating the trade-offs between complexity and number of kernels, providing 19 different kernels (higher than the 12 provided by the CHStone suite) with low memory usage per kernel (up to 64 kiB). Moreover, the complexity per kernel is high enough and the resources occupied low enough to be able to instantiate up to 16 hardware accelerators in the target FPGA. Last but not least, MatchSuite has been used by a higher number of authors when compared to the Rosetta suite, and so it enables more comparisons with the state of the art to evaluate results.

B. Data communication strategies

Data transfers in HW/SW co-processing schemes become the bottleneck in memory-bounded applications. As such, many different analysis and strategies for optimizing memory communication can be found in the literature [6], [7]. Even so, some of these strategies tend to be application-specific [8], making the models and strategies non-generalizable, and others focus on optimizing distributed memory data-paths [9] without optimizing inner-chip communications.

Related to on-chip communications, strategies often focus on improving DMA transfers [10]. In this last case, data coherence and movement management is addressed and modeled, while there is no further study in the modeling of the actual DMA transfer and the impact of using data bursts through DMA. For this purpose, in this paper models of data communications through 16-data burst transfers are analyzed and modeled inside the ARTICo^3 architecture. Moreover, a new feature for ARTICo^3 has been proposed in which the processor is liberated from workload in DMA transfers enabling a double-buffer approach where next-to-go transfers can be previously prepared following a pipelined-like strategy. Finally, this work also serves as an improvement of a previously made architectural analysis where only models of the energy consumption were provided [11].

III. Data-path Modeling

In this section, an analysis of the performance of the communications between virtual user-space memory and the internal memory of the hardware accelerators is presented and discussed. This is of application to hardware accelerators attached to a Linux-based host processor, as is the case of complex infrastructures such as ARTICo^3, but it is also valid for a single accelerator in a Linux-based System on Programmable Chip (SoPC) such as Xilinx Zynq or Intel Stratix 10. In spite of being generalisable to other schemes, the models provided in this work have been obtained for the ARTICo^3 architecture, an accelerator-centric architecture that provides performance scalability on a single SoPC, relying on the application developer to decide how many accelerators to load for a given kernel through the use of Dynamic and Partial Reconfiguration (DPR) [12].

A. Experimental plan

To evaluate the overheads in the the data-path, an analysis of the performance when moving data to and from the memory of the hardware accelerators is provided. The models proposed in this section have been obtained for a Zynq SoPC running a Linux OS for both simple DMA transfers and ARTICo^3 (Linux-based) transfers. The steps followed in the data transfers can be seen in Figure 1, and will be explained in the next subsection. In both send and receive cases, data is moved to/from the processor (the PS, in accordance to the terminology from Xilinx) from/to a memory bank (emulating

hardware accelerator inputs/outputs) instantiated in the PL (the FPGA, also focusing on Xilinx SoPC devices), with data sizes ranging from 4 kiB – 1 MiB and repeating the data transfers 10000 times in order to obtain more accurate average-time values. In both cases data transfers are carried out relying on a kernel module created for the GNU/Linux OS and through Direct Memory Access (DMA), configuring the DMA engine to make data bursts from the PS to the PL.

It is important to highlight the differences between both programming schemes. With regard to simple DMA transfers, 1 to 4 memory banks have been instantiated into the PL and data is transferred directly to those memory banks. When using ARTICo^3, on the other hand, data is not sent to the actual memory of the accelerators (also 1 to 4), but to a shuffler module instantiated in the PL, in charge of distributing the data to the different accelerators.

B. Data-path

The ARTICo^3 framework relies on a list of kernels that have been previously loaded for carrying out data transfers. It searches for the kernel to execute, loads the ports information (inputs and outputs of the kernel) and then transfers the required amount of data to a custom data shuffler implemented in the PL that is in charge of distributing the data between the accelerators, among other functionalities. The steps involved in ARTICo^3 transfers (send and receive between PS/PL), when invoked in a program, are the following:

- Send:
  - Data is copied from user-space memory to DMA memory through the use of a file descriptor and calling the function `memcpy()`, resulting in an elapsed time t_{mcpy-s}.
  - An *ioctl* is invoked to start data transmission from the reserved addresses of the data to the PL through the use of a kernel module. The ioctl can be divided in the following sections:
    - A fixed time, t_{fx-s1}, elapsed from the ioctl call to the invocation of the DMA driver through the function `dmaproxy_ioctl()`.
    - A fixed time, t_{fx-s2}, elapsed from the call of `dmaproxy_ioctl()` until the actual start of the transfer (preparing the DMA configuration or running the DMA microcode, among others).
    - The data transfer, which can be also divided in:
      - Hardware transfer to the data shuffler, t_{hw-s}.
      - Overhead introduced by the OS, t_{oh-s}.
- Receive: Similar to the previous case, but performing an ioctl call before the `memcpy()`. First data is received from the memory of the accelerators to the DMA memory and finally copied to user-space memory.

Grouping both t_{fx-s1} and t_{fx-s2}, the equations will follow the next expressions:

$$t_s = t_{mcpy-s} + t_{fx-s} + t_{hw-s} + t_{oh-s} \quad (1)$$

$$t_r = t_{mcpy-r} + t_{fx-r} + t_{hw-r} + t_{oh-r} \quad (2)$$

Please notice before describing the models that `memcpy()` performance is different on receive and send transmissions, due to the fact that more bandwidth is available during send transactions, if cached data is used. If no cached data is used in send and receive, $t_{mcpy-s} \simeq t_{mcpy-r}$.

All the aforementioned steps can be seen in Figure 1.

C. Models

In this subsection, a mathematical expression is provided to compute each term involved in the previous equations. These terms can be divided into 2 different parts: copying data to the reserved DMA memory (`memcpy()`) and the transfer of that data (`ioctl()`).

- **Models related with copying data to the DMA memory:** The performance of the `memcpy()` instruction is greatly influenced by the use of cached data. Taking this into consideration, 3 different expressions are provided to calculate the time elapsed by calling this function:
  - Cached send:

$$t_{mcpy-s}(ms) = 2.16 * 10^{-6}x \quad (3)$$

  - Cached receive:

$$t_{mcpy-r}(ms) = 4.56 * 10^{-6}x \quad (4)$$

  - Non-cached transfer:

$$t_{mcpy-s/r}(ms) = 6.39 * 10^{-6}x \quad (5)$$

  where x is the total number of bytes to be copied from user-space.
  These 3 expressions correspond to simple `memcpy()` instructions and they can be applied to any kind of Linux application. However, if ARTICo^3 is being used, there is an overhead in the amount of memory used. The consequence is a drop in the performance of cached send transfers, although it is not very significant. ARTICo^3 cached send `memcpy()` performance would follow this other expression:
  - ARTICo^3 cached send:

$$t_{mcpy-s}(ms) = 2.65 * 10^{-6}x \quad (6)$$

- **Models related with data transfers**: these models involve calculating $t_{fx-s}, t_{fx-r}, t_{oh-s}, t_{oh-r}, t_{hw-s}$ and t_{hw-r}. Beginning with fixed elapsed times, the models have resulted to be:

$$t_{fx-s}(ms) = 0.0347 \quad (7)$$

$$t_{fx-r}(ms) = 0.01185 \quad (8)$$

  Next is the actual time elapsed in the hardware transfer. This is the most predictable time, as it is not influenced by the OS. The expressions for 16-data bursts transfers in directly connected accelerators (without involving the shuffler) are the following:

$$t_{hw-s}(ms) = (\frac{data}{16} * 19 + 3n + 1) * \frac{1}{f(kHz)} \quad (9)$$

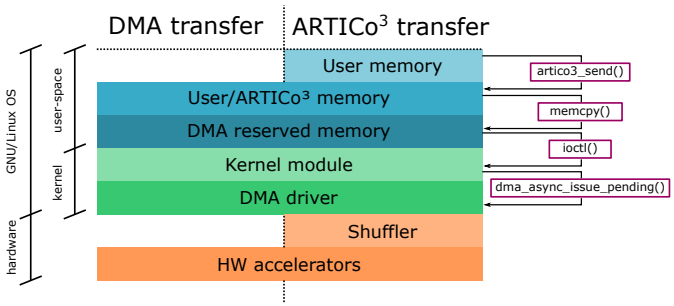


Fig. 1: Data-path on simple and ARTICo^3 DMA transfers

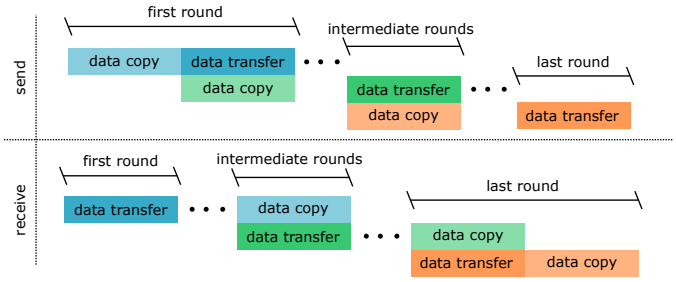


Fig. 2: Double-buffer implementation of ARTICo^3 DMA transfers, send and receive.

$$t_{hw-r}(ms) = (\frac{data}{16} * 22 + 6n - 1) * \frac{1}{f(kHz)} \quad (10)$$

where $data$ is the number of 32-bit data to be transmitted and $n = \lfloor \frac{data}{1024} \rfloor$, consequence of crossing a 4 kiB data addressing boundary. At this point, the DMA engine separates a burst of 16 data into two bursts to adjust the address and to begin addressing again from position 0xYYYYY000.

As mentioned before, ARTICo^3 transfers are not done directly to the memory of the accelerators, but to a data shuffler in charge of managing the data. As a consequence, some latency is introduced and the expressions change, adding some clock cycles per burst. ARTICo^3 hardware data transfer time can be calculated with the following expressions:

$$t_{hw-s}(ms) = (\frac{data}{16} * 29 + 13n + 1) * \frac{1}{f(kHz)} \quad (11)$$

$$t_{hw-r}(ms) = (\frac{data}{16} * 40 + 24n - 1) * \frac{1}{f(kHz)} \quad (12)$$

Finally, the overhead time has resulted to be dependent on the total data to transfer. It can be modeled as a fixed time (nearly the whole time) plus a dependent component:

$$t_{oh-s}(ms) = 0.04751 + (1.072 * 10^{-5})x \quad (13)$$

$$t_{oh-r}(ms) = 0.04956 \quad (14)$$

where, in this case, x corresponds to the total amount of kiB to transfer.

IV. Double-Buffer Approach

In this section, a new double-buffer based approach to data transfers in ARTICo^3 is proposed in order to optimize communications between the PS (host application) and the PL (reconfigurable accelerators).

In the current baseline design of ARTICo^3, when data transfers are being performed, there is a step in which the PS is blocked waiting for the DMA engine to finish the transfer to the hardware accelerators. However, the PS could be liberated during this step to continue with other parts of the application, until the transfer is finished.

The double-buffer approach consists in taking advantage of the liberated CPU time to prepare the next iteration round in a second extra buffer, having both buffers used alternatively. It is important to highlight that the model of computation followed in ARTICo^3 has a policy which resembles to GPUs, in the sense that the input data is split in small units, which are processed by the accelerators in subsequent rounds. The different rounds of execution in send and receive transfers will follow the steps shown in Figure 2.

The pipelined-like approach shown makes transfer time to change in each round from

$$t_{transfer} = t_{mem-cpy} + t_{DMA-transfer} \quad (15)$$

to

$$t_{transfer} = max(t_{mem-cpy}, t_{DMA-transfer}) + t_{db-oh} \quad (16)$$

where t_{db-oh} is the overhead time introduced in the code by the double-buffer approach.

This difference can make execution time in memory-bounded applications to improve in the range of $1.4 - 2\times$, depending on data volume (higher volume, higher improvement). The double buffer minimizes the time the accelerators remain idle waiting for a new data transfer to be produced. Therefore, in computing-bounded applications, performance could receive little to no improvement.

The proposed approach involves changes in the ARTICo^3 runtime library (including the kernel module), but it does not affect user-level programming (i.e., all changes are transparent to the user).

V. Experimental Results

To experimentally evaluate performance improvements, the *MatchSuite* suite has been implemented on the xc7z100ffg900-2 chip (Avnet Zynq MMP board). This benchmark suite comes with a total of 19 kernels to analyze different performance indicators but, due to ARTICo^3 memory limitations in the accelerators, results have been obtained for only 12 of them. It is important to clarify that the selected 12 kernels have been implemented with no modifications, while the other 7 would require modifications or parallelism analysis to adapt them to the ARTICo^3 memory requirements, reason why they were not considered in the experiments.

First, the execution times per round of the 12 kernels have been measured in a bare-metal implementation with an AXI interface between the PL and PS. The results can be seen in

Table: I. Here, it is important to highlight that these executions do not make use of the DMA burst transfers.

The advantage of using the double-buffer comes between successive rounds, as they can be prepared beforehand (data prefetching produced by the double buffer makes no sense if a single round is to be computed). As such, the experimental results correspond to the execution of not just one single round of each kernel, but to 1024 rounds of execution. By using this number of rounds the authors expected the pipelined stage to have reached a saturation of improvement, which was analyzed and corroborated by executing from 1 to 1024 rounds in powers of 2.

In Figure 3, the results are introduced. Note that the comparisons of the single- and double-buffer approach with the bare-metal implementation have been normalized with respect to the bare-metal one due to scale time differences between kernels. The immediate comparison with the bare-metal implementation corresponds with the execution of the kernel in just one hardware accelerator (a). As it can be observed, all ARTICo^3 implementations outperform the bare-metal one with the exception of the md_knn kernel. Even though, this exception is compensated in the double-buffer implementation, which already outperforms the bare-metal one in that kernel. The rest of the figures ((b)-(e)) correspond to one of the scalability possibilities provided by ARTICo^3 (performance scalability), being able to change the amount of hardware accelerators in the target FPGA from 1 to 16 by the use of DPR. As it can be seen, scalability is also improved in the double-buffer implementation, with improvements ranging from 2 to 32% when compared with the single-buffer scalability.

If only execution time is addressed, it is interesting to analyze more deeply Figure 3 (a) and (e), where the bare-metal implementation can be compared with the execution using the same number of accelerators (1) and the maximum (16). The results show that the double-buffer implementation reaches negligible execution times compared with the bare-metal in some kernels, with improvements from $13.7\times$ up to $26.5\times$ with 1 accelerator ($114\times$ up to $318\times$ in the 16 accelerator execution). Furthermore, in more computing-bounded kernels (improvement lower than $5\times$ with 1 accelerator), improvements do not go lower than $1.05\times$ and have an average improvement of $2.51\times$ ($15.7\times$ with 16 accelerators).

The same figures can be used to compare the two versions of ARTICo^3 DMA transfers. Even though there are kernels with little improvement (0.06% minimum) the average amount of improvement goes up to 28% when comparing execution with only one accelerator. Since scalability also goes better in the double-buffer implementation, the mean improvement moves up to 49% with 16 accelerators, a result that really motivates and justifies the use of 2 buffers.

Kernel	Execution Time (ms)
aes	0.888
fft_strided	27.855
gemm_bb	451.794
gemm_ncu	51.753
kmp	49.199
md_grid	58.403
md_knn	5.874
sort_merge	30.622
sort_radix	224.098
spmv_crs	3.887
spmv_ellpack	6.341
viterbi	668.523

TABLE I: Benchmark evaluation results: bare-metal single-word access implementation on Zynq-7000.

VI. Conclusions and Future Work

In this paper, models for ARTICo^3 DMA transfers have been presented together with a proposal of optimization of these transfers. This proposal is based on liberating the CPU while DMA transfers are being carried out, enabling a pipelined-like approach of data transfers between PS and PL by the use of a secondary buffer.

Regarding the models, they have proven to serve as a good reference to calculate the time elapsed in data transfers. Moreover, even though they have been extracted for single-kernel execution with no other application in parallel, they serve as a first step to predict multi-accelerator performance in more complicated models in which the whole system could be taken into consideration, while working on different kernels in parallel.

With these models in mind, a pipelined double-buffer optimization was proposed and developed. The experimental results obtained through the use of the *MatchSuite* benchmark suite have shown the feasibility of this solution. Liberating the CPU of workload has made the new approach outperform the previous polling version in every situation, with improvements in average of 49%. Moreover, these results also suggest that energy consumption should receive a positive impact, which will be studied and analyzed in future work.

Finally, future models are envisioned to increase in complexity when more information is taken into consideration. As such, a future line of work is proposed in which machine learning algorithms are used to autonomously extract these models. This would help on taking more complex decisions at run time regarding the number of accelerators to be used and their configuration, enabling a fine-grain control of the architecture.

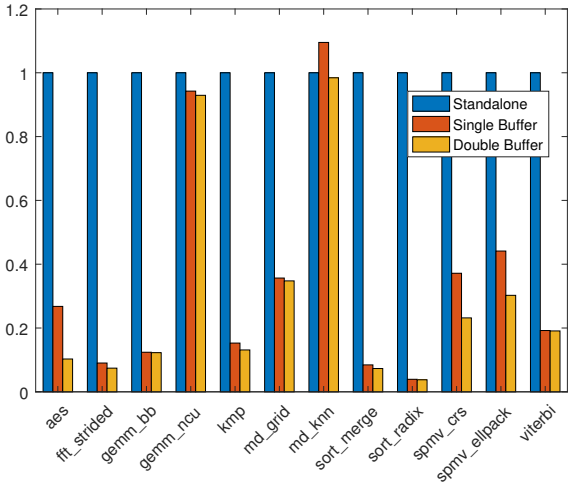
References

[1] T. Todman, G. Constantinides, S. Wilton, O. Mencer, W. Luk, and P. Cheung, *IEE Proceedings - Computers and Digital Techniques*, vol. 152, pp. 193–207(14), March 2005.

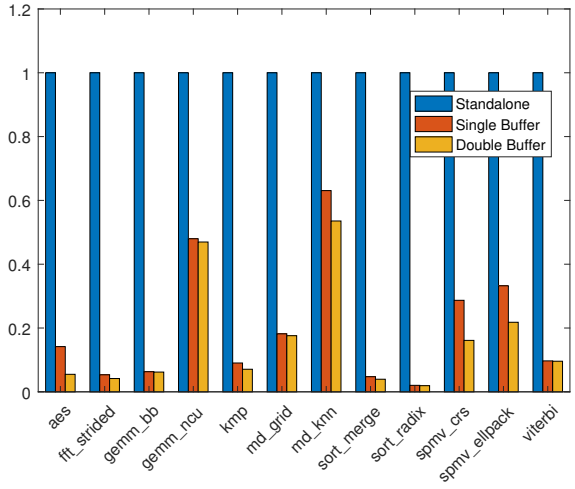
[2] S. Kumar, A. Jantsch, J. . Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, April 2002, pp. 117–124.

[3] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the chstone benchmark program suite for practical c-based high-level synthesis," *Journal of Information Processing*, vol. 17, pp. 242–254, 2009.

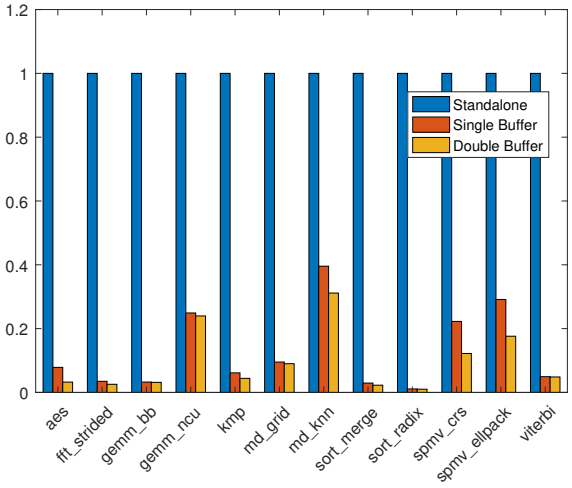
[4] B. Reagen, R. Adolf, Y. S. Shao, G. Wei, and D. Brooks, "Machsuite: Benchmarks for accelerator design and customized architectures," in *2014 IEEE International Symposium on Workload Characterization (IISWC)*, Oct 2014, pp. 110–119.



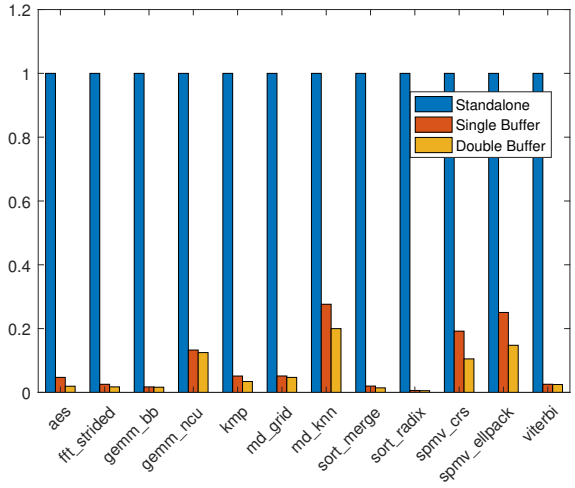
(a) 1 accelerator



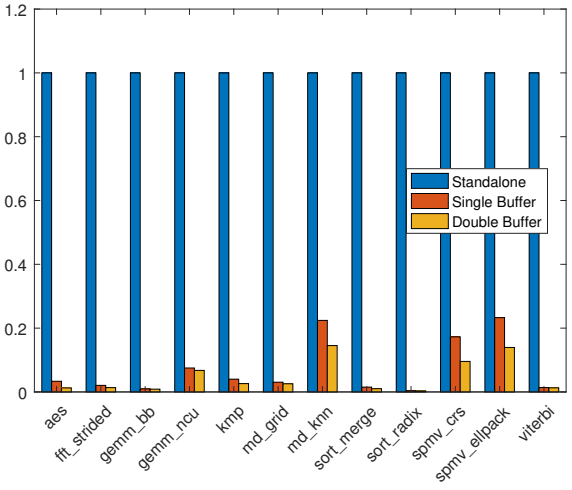
(b) 2 accelerators



(c) 4 accelerators



(d) 8 accelerators



(e) 16 accelerators

Fig. 3: Benchmark evaluation results: Standalone versus ARTICo^3 using single and double buffer. Results are relative to the bare-metal implementation.

[5] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A realistic high-level synthesis benchmark suite for software programmable fpgas," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: ACM, 2018, pp. 269–278. [Online]. Available: http://doi.acm.org/10.1145/3174243.3174255

[6] M. Tahghighi, S. Sinha, and W. Zhang, "Analytical delay model for cpu-fpga data paths in programmable system-on-chip fpga," in *Applied Reconfigurable Computing*, V. Bonato, C. Bouganis, and M. Gorgon, Eds. Cham: Springer International Publishing, 2016, pp. 159–170.

[7] T. Chen and G. E. Suh, "Efficient data supply for hardware accelerators with prefetching and access/execute decoupling," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-49. Piscataway, NJ, USA: IEEE Press, 2016, pp. 46:1–46:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3195638.3195694

[8] A. Rios-Navarro, R. Tapiador-Morales, A. Jimenez-Fernandez, C. Amaya, M. Dominguez-Morales, T. Delbruck, and A. Linares-Barranco, "Performance evaluation over hw/sw co-design soc memory transfers for a cnn accelerator," in *2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO)*, July 2018, pp. 1–4.

[9] S. Shreejith, R. A. Cooke, and S. A. Fahmy, "A smart network interface approach for distributed applications on xilinx zynq socs," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2018, pp. 186–1864.

[10] Y. S. Shao, S. L. Xi, V. Srinivasan, G. Wei, and D. Brooks, "Co-designing accelerators and soc interfaces using gem5-aladdin," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–12.

[11] A. Rodrguez, J. Valverde, C. Castaares, J. Portilla, E. de la Torre, and T. Riesgo, "Execution modeling in self-aware fpga-based architectures for efficient resource management," in *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, June 2015, pp. 1–8.

[12] A. Rodríguez, J. Valverde, J. Portilla, A. Otero, T. Riesgo, and E. De la Torre, "FPGA-Based High-Performance Embedded Systems for Adaptive Edge Computing in Cyber-Physical Systems: The ARTICo^3 Framework," *Sensors*, vol. 18, no. 6, 2018.