Partial Reconfiguration for Design Optimization

Marie Nguyen, Nathan Serafin, and James C. Hoe
Carnegie Mellon University
Pittsburgh, Pennsylvania

***Abstract*—FPGA designers have traditionally shared a similar design methodology with ASIC designers. Most notably, at design time, FPGA designers commit to a fixed allocation of logic resources to modules in a design. At runtime, some of the occupied resources could be left under-utilized due to hard-to-avoid sources of inefficiencies (e.g., operation dependencies, unbalanced pipelines). With partial reconfiguration (PR), FPGA resources can be re-allocated over time. Therefore, using PR, a designer can attempt to reduce under-utilization with better area-time scheduling.**

In this paper, we offer definitions, insights, and equations to explain when, how, and why PR-style designs can improve over the performance-area Pareto front of ASIC-style designs (without PR). We first introduce the concept of area-time volume to explain why PR-style designs can improve upon ASIC-style designs. We identify resource under-utilization as an opportunity that can be exploited by PR-style designs. We then present a first-order analytical model to help a designer decide if a PR-style design can be beneficial. When it is the case, the model points to the most suitable PR execution strategy and provides an estimate of the improvement. The model is validated in a case study. [1]

I. Introduction

Motivations. Today, with growing emphasis on deploying Field Programmable Gate Arrays (FPGAs) for computing, we are starting to see FPGAs' reprogrammability being recognized as a deciding feature in selecting FPGAs over ASICs [1]. Yet, partial reconfiguration (PR), which allows parts of an FPGA to be reconfigured at millisecond timescales, remains an under-appreciated capability.

This paper explores the questions of when, how, and why FPGA designers should consider using PR. The discussions in this paper focus on the use of PR in challenging design scenarios that have to deliver required performance under strict area, cost, power, and energy constraints (e.g., [2], [3]). This work is particularly relevant to AI-driven applications at the Edge (e.g., [4], [5], [2], [3]) that (1) are deployed on low-end FPGAs due to cost, power, and size concerns, and (2) need to accelerate many compute intensive tasks with stringent latency or throughput requirements ([2], [6], [7]).

Shortcomings of ASIC-Style Designs. To accelerate these constrained applications on the FPGA, designers commit, at design time, to a fixed allocation of logic resources to modules. We refer to this design as an ASIC-style design. At runtime, some of the occupied resources could be left idle or under-utilized due to hard-to-avoid sources of inefficiencies (e.g., operation dependencies, unbalanced pipelines) which may occur even in a highly-optimized design. Under-utilization may result in (1) the design not running at the desired performance given an area budget, or (2) the design running at the desired performance but being too big to fit in the given area.

[1]Refer to the long paper version on arXiv for a full evaluation.

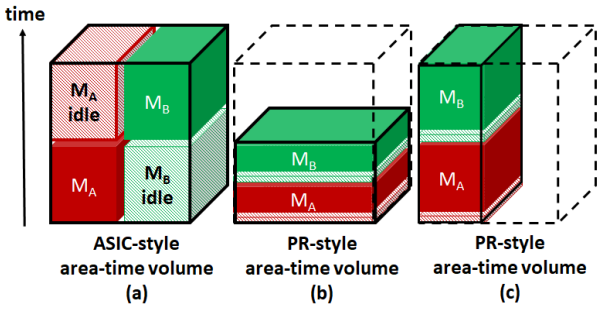

Fig. 1: In an ASIC-style design, logic resources that are inactive still occupy the fabric. In a PR-style design, under-utilization can be reduced with better area-time scheduling.

PR-Style Designs to Reduce Under-Utilization. Using PR, a designer can attempt to reduce under-utilization by changing the allocation of resources over time. In this paper, we identify under-utilization of resources as an opportunity that can be exploited by PR-style designs to improve upon ASIC-style designs. We refer to a PR-style design as a design in which *logic resources are allocated to different modules of one design over time. In return, a PR-style design may be faster and/or smaller than an ASIC-style design* (illustration in Figure 1).

This work: when, how and why PR. To address the questions of when, how, and why PR, this paper develops a set of PR execution strategies applicable to a range of non-trivial applications (e.g., video analytics/image processing pipelines, feed-forward neural networks). An application consists of a set of tasks, and each task is accelerated by a hardware module. Modules can be dependent, execute concurrently, and have multiple implementation variants with different performance-area trade-offs. Dependent modules share data either through (1) external memory or (2) on-chip memory. The paper proposes a first-order analytical model to help a designer (1) determine a suitable PR execution strategy and (2) analyze the throughput and latency of ASIC-style and PR-style designs. The model enables quick exploration of the design space to help decide if a PR-style design can be beneficial for a given problem. The effectiveness of this model is examined in a compute-bound case study.

The contributions of this paper are:

- developing a set of PR allocation and scheduling strategies for practical design scenarios
- developing a first-order performance model to estimate

ASIC-style and PR-style designs' performance

- demonstrating the effectiveness of the performance model with a case study of an implemented design.

II. Background and Related Work

Partial Reconfiguration. When using PR, the FPGA fabric is divided into a non-reconfigurable region (containing the I/O infrastructure) and PR regions that can be reprogrammed individually at runtime. Each PR region can be reprogrammed at runtime with partial bitstreams built for this region at design time. When loading bitstreams from on-board DRAM, the time to load a PR region (PR time) is a function of the bitstream size, e.g., approximately 453 MB/sec on an Ultrascale+ device through the processor configuration access port (PCAP).

Applications of PR Today. Many academic projects have explored the potential of using PR (e.g., [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]). Commercially, PR has been mainly used in a "role-and-shell" approach ([1], [21]). A static shell design provides I/O and isolation while independent designs with different functionalities, or roles, can be loaded as required (e.g., [1], [21]). The different role designs reuse the same logic resources over time. However, each role is still an ASIC-style design. This paper does not focus on using PR in a role-and-shell approach.

PR-Style Benefits. In a PR-style design, the designer decides how the FPGA is divided into PR regions and when/which reconfigurations are needed during the design's execution. For instance, in [22], to accelerate a vision processing pipeline, a PR region is reconfigured every few milliseconds with different pipeline stages.

For domain-specific applications, other prior works have exploited under-utilization in ASIC-style designs, and have shown that using PR can provide area ([23]), performance ([24], [25], [26]), power/energy ([27], [28]), and compilation time reduction ([29]) benefits. For instance, in adaptive ([30]) or cloud computing applications ([31], [32], [33]), multiple modes or implementation variants exist for a module but only one is needed at a time depending on the context. Instead of mapping all variants of a module in an ASIC-style design, only one variant is reprogrammed on the fabric at a time.

Scheduling for PR-Style. A vast body of work on FPGA OSes ([34], [35], [36], [37], [38]) and on FPGA virtualization ([39], [33], [31]) has focused on the theory of spatial and temporal sharing, mechanisms for task preemption, or hardware and software task scheduling ([40]). Mostly, these works share the common goal of maximizing resource utilization to improve throughput, and often assume that PR time is negligible compared to compute, and/or that tasks are independent.

Building on top of prior work, this paper introduces the concept of area-time volume, and also gives practical examples of when PR can be beneficial. We consider both throughput and latency, which is the metric of interest for an emerging class of Edge applications that have tolerance for 100 ms-response time, and that could benefit from FPGA acceleration ([6], [7]). We also account for cases where PR time can be equal to or greater than compute time.

III. When and how can PR help?

In this section, we use an idealized and simplified example to develop the intuitions behind when and how PR-style designs can be faster or smaller than ASIC-style designs.

A. Simplified Execution Model

We consider an application with two dependent tasks, task_A and task_B; task_B can start only after task_A is finished. Each task runs once per execution of the application. The latency of the application is the sum of the two dependent tasks' latencies. Multiple implementation module variants exist for task_A and task_B and are characterized by the latency function $Lat_i()$. $Lat_i(a)$ is the latency achieved by the module variant for task_i using a logic resources. For a given task_i, larger variants have lower latency, $Lat_i(a) < Lat_i(b)$ if $a > b$. In this example, we assume that performance scales linearly with area. The complete model presented in next section accounts for other type of scaling, e.g., sub-linear.

B. ASIC-Style Design

Consider two common design objectives: (1) minimize latency given an area budget, or (2) minimize area given a latency upper bound. For simplicity, assume $Lat_A(a) = Lat_B(a)$ for any a. In that case, to achieve optimality in either objective, the total logic resources, A_{total}, must be equally divided between task_A and task_B's modules ($A_A = A_B = 0.5A_{\text{total}}$). The latency of the application is $2Lat_{A/B}(0.5A_{\text{total}})$. Solving either optimization scenarios for different latency or area targets will produce a set of ASIC-style implementations that trade off latency against logic resources.

The above scenario for ASIC-style design is shown in Figure 1.a. The FPGA fabric area is 100% occupied by the modules for task_A and task_B. However, due to the module dependency, only one of the two modules is active at a time.

C. PR-Style Design

In a PR-style design, to minimize latency given the same area budget, we can allocate the entirety of A_{total} to a module for task_A first and then to task_B (Figure 1.b). By doing so, the PR-style design's latency is reduced as both modules now run faster using all of the resources available. To maintain the same latency using half the resources, we can allocate $0.5A_{\text{total}}$ to a module for task_A first and then to task_B (Figure 1.c). In Figures 1.b and c, both PR-style designs fit into smaller area-time volumes than the ASIC-style design even when accounting for the non-zero delay to perform PR when switching between modules.

D. Opportunities for Improvement by PR

In this example, resource under-utilization due to module dependencies cannot be eliminated without changing the initial algorithm or implementation. In practice, under-utilization can arise in other forms in ASIC-style designs. In our simplified example, we assume that module variants exist for any amount of resources. However, module variants for a task only exist at certain performance/resource combinations. The modules

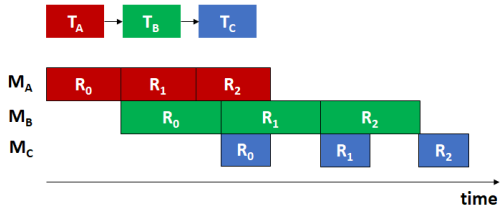

Fig. 2: Example timeline of an application with three dependent tasks accelerated by modules M_A, M_B, and M_C.

selected to fit an area budget in an ASIC-style design may not sum up perfectly to use all resources. Further, when the modules of task_A and task_B are executed in a pipelined fashion to improve the throughput of many independent executions, it may not be possible to find variants with equal throughput for the two tasks; in the resulting unbalanced pipeline, a too-fast stage has to stop or slow down to wait for the other stage. A more subtle example exists when implementing a generic engine capable of accelerating different algorithms or neural networks. This generalized engine consists of a superset of features to accommodate all possibilities but only a subset of features is needed at a time (e.g., NPU [41], DPU [42]). A PR-style design could remove this type of inefficiencies.

IV. Analytical Model

In this section, we present our model and discuss the additional memory requirements of a PR-style design.

A. Overview

Optimization Goals. To derive our performance model, we consider the problem of maximizing an application's performance given an area budget.

- *minimize the application's latency given an area budget* A. We label this problem as **min L given A**.
- *maximize the application's throughput given an area budget* A. We label this problem as **max T given A**.

Execution Model. In this section, we consider an application with N dependent tasks; each task is accelerated by a module. Dependent modules share data either through external or on-chip memory depending on data size. Though our discussion focuses on applications with dependent tasks, our model also applies if tasks are independent. We define I as the set of subscripts for tasks in the application. A single start-to-finish execution of a module is referred to as a run. If an application requires multiple independent runs, modules can execute concurrently. Figure 2 illustrates this execution model. The example application consists of three dependent tasks task_A, task_B, and task_C accelerated by three modules. In this application, each module needs to complete three runs R_0, R_1, and R_2. Modules execute concurrently to complete the runs as quickly as possible, subject to the dependency constraints.

We consider two performance metrics, latency and throughput. Latency is defined as the start-to-finish time required for all modules accelerating an application to complete one run (including I/O time for data read and write and compute time).

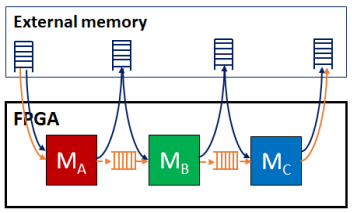

Fig. 3: In an ASIC-style design, dependent modules share data through either external (blue) or on-chip memory (orange) depending on data size.

Throughput is defined as the number of runs completed per unit time in steady-state.

Performance-Area Trade-offs. For each module, a finite set of implementation variants exists. A variant accelerating task_i is characterized by its area a_i, its latency $Lat_i(a_i)$, and its throughput $Tput_i(a_i)$ as functions of area. We assume that Lat_* and $Tput_*$ are monotonically increasing functions but make no further assumption on their shape, e.g., performance can scale sub-linearly or lineary with area.

PR-Style Design Considerations. We define $Time_{PR}(a)$ as the time to reconfigure a PR region of size a, and assume that PR time is proportional to the PR region size.

B. ASIC-Style

We first derive the equations for the ASIC-style design that are applicable whether dependent modules share data through external or on-chip memory (Figure 3). In both cases, the number of buffers required to hold intermediate data is $N+1$.

Min L Given A. Let $Lat_{\mathsf{Asic}}(A)$ be the latency of the ASIC-style design given A resources.

$$Lat_{\mathsf{Asic}}(A) = \sum_{i \in I} Lat_i(a_i), \sum_{i \in I} a_i \leq A \tag{1}$$

Max T Given A. Let $Tput_{\mathsf{Asic}}(A)$ be the throughput of the ASIC-style design given A resources.

$$Tput_{\mathsf{Asic}}(A) = \min(\{Tput_i(a_i) \mid i \in I\}), \sum_{i \in I} a_i \leq A \tag{2}$$

C. Ignoring PR Time: PR-Style Performance Bounds

Ignoring PR time, we first derive the lower and upper bounds on the latency and throughput, respectively, achievable by any PR-style design presented in the next subsections. The simplest and most efficient execution strategy is to schedule tasks serially on one PR region. Each module runs once before the PR region is reconfigured with the next module. In the best-case scenario, the PR region is of size A and the highest performance variant using A resources exists for all modules.

Min L Given A. Let $Lat_{\mathsf{PR,1,min}}(A)$ be the lower bound on latency for the PR-style design with one PR region.

$$Lat_{\mathsf{PR,1,min}}(A) = \sum_{i \in I} Lat_i(A) \tag{3}$$

Max T Given A. Let $Tput_{\mathsf{PR,1,max}}(A)$ be the upper bound on throughput for the PR-style design with one PR region.

$$Tput_{\mathsf{PR,1,max}}(A) = \frac{1}{\sum_{i \in I} \frac{1}{Tput_i(A)}} \tag{4}$$

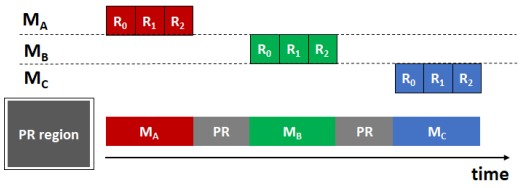

Fig. 4: Example of serialized execution in a PR-style design with one PR region when batching ($B = 3$).

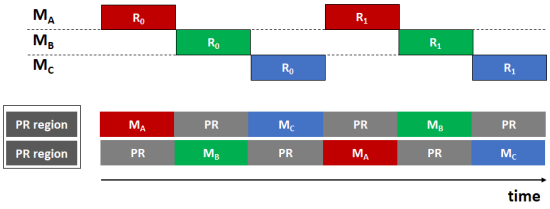

Fig. 5: Interleaved execution on two PR regions. PR time can be hidden by overlapping compute and reconfiguration.

D. Including PR Time: Serialized Execution on one PR Region

When accounting for PR time and scheduling tasks serially on one PR region, each module runs once before the PR region is reconfigured with the next module. Given N tasks, the PR region is reconfigured N times. Compute and reconfigurations are serialized.

Min L Given A. Let $Lat_{\mathsf{PR},1}(A)$ be the latency of the PR-style design with one PR region.

$$Lat_{\mathsf{PR},1}(A) = \sum_{i \in I} Lat_i(a_i) + N \times Time_{PR}(A) \tag{5}$$

Scheduling tasks serially on one PR region of the largest size may not result in the design's minimum latency. Though using larger variants leads to a decrease in compute time, it also has the effect of increasing PR time, which may offset the speedup benefit of larger variants. In the next subsection, we discuss a scheduling alternative where compute and reconfigurations are overlapped.

Max T Given A: Batching to Amortize PR Time. Let $Tput_{\mathsf{PR},1}(A)$ be the steady-state throughput of the PR-style design with one PR region.

$$Tput_{\mathsf{PR},1}(A) = \frac{1}{\sum_{i \in I} \frac{1}{Tput_i(a_i)} + N \times Time_{PR}(A)} \tag{6}$$

If PR time is non-trivial compared to compute time, we can amortize PR time by executing each module B times (i.e. batching B runs) before reconfiguring the PR region (Figure 4). Let $Tput_{\mathsf{PR},1}(A)$ be the steady-state throughput of the PR design with one PR region when batching runs.

$$Tput_{\mathsf{PR},1}(A) = \frac{B}{\sum_{i \in I} \frac{B}{Tput_i(a_i)} + N \times Time_{PR}(A)} \tag{7}$$

Batching allows us to reduce the ratio of total PR time to total compute time at a greater resource cost to buffer intermediate results. Given enough buffering capacity, PR time can be almost totally amortized for large enough B.

E. Including PR Time: Special Cases

Min L Given A: Interleaved Execution on Two PR regions. When optimizing for latency, interleaving task execution on multiple PR regions allows us to overlap reconfigurations and compute to hide PR time, which may result in better latency than serializing task execution on one PR region. Figure 5 shows an example of interleaved execution for $k = 2$. In this example, $Time_{PR}(A/2) = Lat_i(a_i), \forall i \in I$. By overlapping compute and reconfigurations, PR time is completely hidden. Having $k > 2$ may be beneficial provided that multiple PR regions can be reconfigured simultaneously. Simultaneous reconfiguration of multiple PR regions is not supported from a user standpoint using current FPGA tools and PR flow. In this paper, we only consider the case where $k = 2$, and define $Lat_{\mathsf{PR},2}(A)$ as the latency of the PR-style design with two PR regions.

$$Lat_{\mathsf{PR},2}(A) = \sum_{i \in I} \max(Time_{PR}(A/2), Lat_i(a_i)) \tag{8}$$

Max T Given A: Serialized Execution on k PR regions. When optimizing for throughput, it is generally preferable to choose the smallest k to reduce a design's complexity in terms of buffering management since each PR region requires its own intermediate buffer. A k-PR region solution should be considered when appropriately large module variants are not available for all modules in a single PR region solution.

When having multiple PR regions executing in parallel (similar to k-way SIMD), task execution can be serialized on each PR region of size A/k. On each PR region, each module runs once or multiple times before the PR region is reconfigured. Let $Tput_{\mathsf{PR},1}(A/k)$ be the steady-state throughput of a single PR region of size A/k and $Tput_{\mathsf{PR},k}(A)$ be the steady-state throughput of the PR-style design with k PR regions. Assuming that k reconfigurations can occur simultaneously,

$$Tput_{\mathsf{PR},k}(A) = k \times Tput_{\mathsf{PR},1}(A/k) \tag{9}$$

As explained previously, only one reconfiguration can happen at a time using current tools. The above throughput can still be achieved by offsetting the start of compute on each PR region by a sufficient number of PR times to ensure that two PR regions are not reconfigured simultaneously.

F. Memory Requirements in PR-style designs

Min L Given A. Whenever optimizing for latency, the same amount of memory capacity and bandwidth is needed for ASIC-style and PR-style designs (modules do not execute concurrently and only complete a single run).

Max T Given A. When optimizing for throughput and with batching, each PR region requires two intermediate buffers to hold the intermediate input and output data (each of size B). An upper bound on the extra memory bandwidth can be determined by considering the bandwidth required (for read and write) by the variant with the highest throughput.

TABLE I: Resource utilization, average memory bandwidth, and throughput of the ASIC-style design and the module variants used post place & route on the Ultra96 v2 board at 150 MHz.

	Module variants			ASIC-style		
	hog	**cnn**	**lstm**	**I/O Infrastructure**	**Modules**	**Total**
LUT	15,495 (22%)	14,614 (20.7%)	7715 (10.9%)	6082 (8.6%)	37,824 (53.6%)	43,906 (62.2%)
BRAM36k	34 (15.7%)	92 (42.6%)	80.5 (37.3%)	0	206.5 (95.6%)	206.5 (95.6%)
DSP	64 (17.8%)	10 (2.8%)	7 (1.9%)	0	81 (23%)	81 (23%)
Memory bandwidth (MB/s)	23.6	42.7	3.3	N/A	N/A	64.6
Throughput (fps)	30	16	271	N/A	N/A	16

TABLE II: Resource utilization of the PR-style design $\mathbf{P_1}$ post place & route on the Ultra96 v2 board at 150 MHz. Most resources are spent for compute.

	I/O infrastructure	**PR region**	**Total**
LUT	3366 (4.8%)	61,920 (87.8%)	65,286 (92.5%)
BRAM36k	0	198 (91.7%)	198 (91.7%)
DSP	0	288 (80%)	288 (80%)
PR time (ms)	N/A	12	N/A

TABLE III: Resource utilization and throughput of the variants used in $\mathbf{P_1}$.

	hog	**cnn**	**lstm**	**stereo**	**flow**	**viola**
LUT	55,635	27,573	47,745	51477	40,509	42,283
BRAM 36k	109	180	144	96.5	195	91.5
DSP	114	11	13	0	49	101
Throughput (fps)	116	32	2.1k	240	180	41.3

If the intermediate buffers are stored in external memory due to a lack of on-chip memory, the designer needs to ensure that the system has enough external memory bandwidth for the PR-style design to operate at the desired throughput. For the type of applications targeted in this paper, the requirements in external memory capacity and bandwidth do not exceed tens of MBs and hundreds of MB/s, respectively.

V. Results

In this section, we explore the **Max T Given A** problem on a case study. Three dependent tasks are accelerated by a hog, a cnn and an lstm module. We show that (1) our first-order model allows us to accurately estimate a design's performance, and (2) PR-style designs improve resource utilization, and, therefore, performance over ASIC-style designs.

ASIC-style. Table I shows the resource utilization and the throughput of the ASIC-style design and the module variants used. The ASIC-style design's throughput is equal to 16 fps and is limited by the throughput of the slowest module (cnn). The ASIC-style design has modules with mismatched throughput since the amount of computation per frame for the lstm variant is much less than the two other modules.

PR-style. Based on our analysis, batched execution on a single PR region solution $\mathbf{P_1}$ should improve performance. Table II reports the resource utilization of $\mathbf{P_1}$. The time to reconfigure the PR region through the PCAP when PR bitstreams are stored in external DDR is 12 ms (PR bitstreams of 5.5 MB for $\mathbf{P_1}$). We use one ARM core to manage the operation of the fabric at runtime (reconfiguration of the PR regions and module execution).

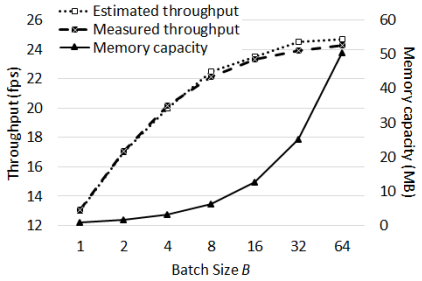

Fig. 6: Throughput and memory capacity of $\mathbf{P_1}$ vs. B for the case study.

Figure 6 shows the estimated and measured throughput, and the intermediate buffering capacity required for $\mathbf{P_1}$ vs. batch size B. We use equation 7, measured throughput variants (Table III) and PR time (Table II) to compute these estimations. We observe that (1) as predicted by the model, when B increases, PR time gets amortized, but with diminishing return when $B \geq 32$. (2) For all B, the estimated and measured throughput match within 2.35%. (3) At $B = 64$, the throughput of the PR-style design is 24.2 fps, which represents a 54.4% improvement over the ASIC-style design. (4) Intermediate buffering capacity linearly increases with B, and is equal to 50.3 MB for $B = 64$. The intermediate buffers are stored in external memory (2GB external DDR available on the Ultra96). The peak external memory bandwidth (read and write) requirement for $\mathbf{P_1}$ is 91.2 MB/s due to the hog module. This represents a 41.2% increase over the ASIC-style design which needs on average 64.6 MB/s (Table I).

VI. Conclusion

In this paper, we develop a set of PR execution strategies for non-trivial design scenarios, and a performance model to quickly and accurately estimate the relative merits of ASIC-style and PR-style designs. We validate our first-order model in a study application that serves as a practical example of ASIC-style design with under-utilization.

VII. Acknowledgments

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. We thank Xilinx and Intel for their FPGA and tool donations.

References

[1] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou,

and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1–13, Oct 2016.

[2] S. Wang, C. Zhang, Y. Shu, and Y. Liu, "Live video analytics with fpga-based smart cameras," in *Workshop on Hot Topics in Video Analytics and Intelligent Edges (HotEdgeVideo)*, October 2019.

[3] N. S. Kim and P. Mehra, "Practical near-data processing to evolve memory and storage devices into mainstream heterogeneous computing systems," in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), pp. 22:1–22:4, ACM, 2019.

[4] Microsoft, "Live video analytics," 2012.

[5] Megh Computing, 2019.

[6] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, (New York, NY, USA), pp. 14:1–14:14, ACM, 2017.

[7] W. Zhang, S. Li, L. Liu, Z. Jia, Y. Zhang, and D. Raychaudhuri, "Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1270–1278, April 2019.

[8] M. Majer, J. Teich, A. Ahmadinia, and C. Bobda, "The Erlangen Slot Machine: A Dynamically Reconfigurable FPGA-based Computer," *J. VLSI Signal Process. Syst.*, vol. 47, pp. 15–31, Apr. 2007.

[9] X. Li, X. Wang, F. Liu, and H. Xu, "Dhl: Enabling flexible software network functions with fpga acceleration," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1–11, July 2018.

[10] C. Dennl, D. Ziener, and J. Teich, "On-the-fly composition of fpga-based sql query accelerators using a partially reconfigurable module library," in *2012 IEEE 20th International Symposium on Field-Programmable Custom Computing Machines*, pp. 45–52, April 2012.

[11] E. Caspi, M. Chu, R. Huang, J. Yeh, J. Wawrzynek, and A. DeHon, "Stream computations organized for reconfigurable execution (score)," in *Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, FPL '00, (Berlin, Heidelberg), p. 605–614, Springer-Verlag, 2000.

[12] B. A. Farisi, K. Heyse, and D. Stroobandt, "Reducing the overhead of dynamic partial reconfiguration for multi-mode circuits," in *2014 International Conference on Field-Programmable Technology (FPT)*, pp. 282–283, Dec 2014.

[13] C. Huriaux, O. Sentieys, and R. Tessier, "Fpga architecture support for heterogeneous, relocatable partial bitstreams," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–6, Sep. 2014.

[14] S. Bhandari, S. Subbaraman, S. Pujari, F. Cancare, F. Bruschi, M. D. Santambrogio, and P. R. Grassi, "High speed dynamic partial reconfiguration for real time multimedia signal processing," in *2012 15th Euromicro Conference on Digital System Design*, pp. 319–326, Sep. 2012.

[15] M. Dyer, C. Plessl, and M. Platzner, "Partially reconfigurable cores for xilinx virtex," in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream* (M. Glesner, P. Zipf, and M. Renovell, eds.), (Berlin, Heidelberg), pp. 292–301, Springer Berlin Heidelberg, 2002.

[16] H. Omidian and G. G. Lemieux, "Software-based dynamic overlays require fast, fine-grained partial reconfiguration," in *Proceedings of the 10th International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, HEART 2019, (New York, NY, USA), Association for Computing Machinery, 2019.

[17] J. Goeders, T. Gaskin, and B. Hutchings, "Demand driven assembly of fpga configurations using partial reconfiguration, ubuntu linux, and pynq," in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 149–156, April 2018.

[18] L. Gong and O. Diessel, "Resim: A reusable library for rtl simulation of dynamic partial reconfiguration," in *2011 International Conference on Field-Programmable Technology*, pp. 1–8, Dec 2011.

[19] M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An fpga run-time system for dynamical on-demand reconfiguration," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pp. 135–, April 2004.

[20] N. Thomas, A. Felder, and C. Bobda, "Adaptive controller using runtime partial hardware reconfiguration for unmanned aerial vehicles (uavs)," in *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–7, Dec 2015.

[21] Amazon, "Amazon EC2 F1 Instances."

[22] M. Nguyen and J. C. Hoe, "Time-shared execution of realtime computer vision pipelines by dynamic partial reconfiguration," in *28th International Conference on Field Programmable Logic and Applications, FPL 2018, Dublin, Ireland, August 27-31, 2018*, pp. 230–234, 2018.

[23] C. Claus, W. Stechele, and A. Herkersdorf, "Autovision – a run-time reconfigurable mpsoc architecture for future driver assistance systems (autovision – eine zur laufzeit rekonfigurierbare mpsoc architektur für zukünftige fahrerassistenzsysteme)," vol. 49, pp. 181–, 05 2007.

[24] D. Koch and J. Torresen, "FPGASort: A High Performance Sorting Architecture Exploiting Run-time Reconfiguration on Fpgas for Large Problem Sorting," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '11, (New York, NY, USA), pp. 45–54, ACM, 2011.

[25] J. Arram, W. Luk, and P. Jiang, "Ramethy: Reconfigurable Acceleration of Bisulfite Sequence Alignment," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, (New York, NY, USA), pp. 250–259, ACM, 2015.

[26] A. Sudarsanam, R. Barnes, J. Carver, R. Kallam, and A. Dasu, "Dynamically reconfigurable systolic array accelerators: A case study with extended kalman filter and discrete wavelet transform algorithms," *IET Computers Digital Techniques*, vol. 4, pp. 126–142, March 2010.

[27] J. Noguera and I. O. Kennedy, "Power reduction in network equipment through adaptive partial reconfiguration," in *2007 International Conference on Field Programmable Logic and Applications*, pp. 240–245, Aug 2007.

[28] M. Nguyen, R. Tamburo, S. Narasimhan, and J. C. Hoe, "Quantifying the benefits of dynamic partial reconfiguration for embedded vision applications," in *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 129–135, Sep. 2019.

[29] S. Ma, Z. Aklah, and D. Andrews, "Just in time assembly of accelerators," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, (New York, NY, USA), p. 173–178, Association for Computing Machinery, 2016.

[30] V. Kizheppatt and S. Fahmy, "Efficient region allocation for adaptive partial reconfiguration," pp. 1–6, 12 2011.

[31] S. A. Fahmy, K. Vipin, and S. Shreejith, "Virtualized fpga accelerators for efficient cloud computing," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 430–435, Nov 2015.

[32] F. Chen, Y. Shan, Y. Zhang, Y. Wang, H. Franke, X. Chang, and K. Wang, "Enabling fpgas in the cloud," in *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, (New York, NY, USA), pp. 3:1–3:10, ACM, 2014.

[33] S. Byma, J. G. Steffan, H. Bannazadeh, A. L. Garcia, and P. Chow, "FPGAs in the Cloud: Booting Virtualized Hardware Accelerators with OpenStack," in *2014 IEEE 22nd Annual International Symposium on Field-Programmable Custom Computing Machines*, pp. 109–116, May 2014.

[34] A. Khawaja, J. Landgraf, R. Prakash, M. Wei, E. Schkufza, and C. J. Rossbach, "Sharing, protection, and compatibility for reconfigurable fabric with amorphos," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, (Carlsbad, CA), pp. 107–127, USENIX Association, Oct. 2018.

[35] W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews, "Hthreads: A computational model for reconfigurable devices," in *2006 International Conference on Field Programmable Logic and Applications*, pp. 1–4, Aug 2006.

[36] A. Agne, M. Happe, A. Keller, E. Lübbers, B. Plattner, M. Platzner, and C. Plessl, "Reconos: An operating system approach for reconfigurable computing," *IEEE Micro*, vol. 34, pp. 60–71, Jan 2014.

[37] H. K.-H. So and R. Brodersen, "A unified hardware/software runtime environment for fpga-based reconfigurable computers using borph," *ACM Trans. Embed. Comput. Syst.*, vol. 7, pp. 14:1–14:28, Jan. 2008.

[38] K. Fleming, H. Yang, M. Adler, and J. Emer, "The leap fpga operating system," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–8, Sep. 2014.

[39] A. Vaishnav, K. D. Pham, D. Koch, and J. Garside, "Resource elastic virtualization for fpgas using opencl," in *2018 28th International Con-*

ference on Field Programmable Logic and Applications (FPL), pp. 111–1117, Aug 2018.

[40] S. Banerjee, E. Bozorgzadeh, and N. Dutt, "Physically-aware hw-sw partitioning for reconfigurable architectures with partial dynamic reconfiguration," in *Proceedings. 42nd Design Automation Conference, 2005.*, pp. 335–340, June 2005.

[41] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, M. Ghandi, S. Heil, P. Patel, A. Sapek, G. Weisz, L. Woods, S. Lanka, S. K. Reinhardt, A. M. Caulfield, E. S. Chung, and D. Burger, "A configurable cloud-scale dnn processor for real-time ai," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, June 2018.

[42] Xilinx, *Zynq DPU v3.1*, 2019.