Enabling Hardware Design Tradeoffs at Runtime with Dynamically Composable Designs for FPGAs

Shashank Obla
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
sobla@andrew.cmu.edu

***Abstract*—The traditional static ASIC-style approach to FPGA design is incompatible with FPGA deployment and the dynamic software paradigm in servers. While balancing current FPGA limitations with anticipated Partial Reconfiguration (PR) support, this work proposes a framework that (a) promotes a service-oriented design style, enabling designs to be composed of reusable modules for faster development and (b) supports dynamic composability of modules and their variants using PR enabling runtime tradeoff resolution. A proof-of-concept using a service-oriented implementation of BFS targeting (a) caching and (b) graph representations, shows that such a framework can, without compromising performance, achieve lower resource utilization as a function of inputs and FPGA load at runtime.**

I. Introduction

Overview: Due to the stagnation of technology scaling, the ever-growing demand for faster applications has led to the increasing interest in technologies that allow specialization. FPGAs, being one such resource, are approaching ubiquity in the computing space with their application to the acceleration of datacenter functions and applications. While keeping up with the rapid development of new algorithms, sharing, protection and quality of service guarantees demand responsiveness to the dynamic circumstances found in server systems. Many of these challenges are not newly discovered and have been studied extensively in processor and GPU based server systems, but need to be re-evaluated for FPGA-centric systems with tradeoffs at different proportions and priorities.

Traditional FPGA design is heavily based on static ASIC-style design and toolflow. This approach leads to a complex and lengthy design process with long development cycles, and the lack of a standardized specification or software abstraction exacerbates design portability. ASIC-style thinking also brings with it the inefficiency of over-provisioning each design to balance all tradeoffs at design time, leaving significant runtime slack waiting to be exploited. Partial Reconfiguration (PR), i.e., runtime reconfiguration of a subset of the fabric while the rest of the fabric continues uninterrupted, is vital in enabling FPGAs to face these challenges.

Background: Recent approaches such as [1] provide OS like support for FPGAs under current technological constraints but are not future proof and only work under assumptions not characteristic of server application load. [2] develops a system for streaming applications using PR to break away from the ASIC mentality, but needs to be generalized for deployment at datacenter scale. [3] proposes a service-oriented architecture for FPGAs, providing the needed discipline to ease FPGA design enabling reuse and composability at design time.

Exploiting Dynamism: This work proposes a framework to allow dynamic composability of modules using PR, breaking away from the ASIC mentality, which allows for better tradeoff resolution based on runtime constraints, such as inputs and co-scheduled applications, rather than static constraints. The framework is based on the service-oriented modularity [3] allowing for module variants to change functionality or tune performance, energy and area through parametrization or completely new algorithms and implementations. This design style also decouples efficient hardware module design from system design making FPGAs more accessible to software developers and reducing overall design time.

The benefits of this dynamic modularity are exemplified using a service-oriented implementation of BFS at various levels in the design hierarchy. The proof-of-concept is implemented on a server-grade FPGA and includes a shell to interface with peripherals and enable PR. Limited PR support is added to the services framework to evaluate the example and also study the challenges in introducing structure in a fabric built for flexibility, such as floorplanning, and context saving. Tradeoffs in caching and graph representations show that such a framework can react on an input-to-input basis to save area, power and memory without sacrificing performance.

II. Background

This section describes three previous works which, along with the ideas introduced in this paper, form the basis for the framework described in Section III.

A. FPGAs in the Cloud

FPGAs in servers are a major departure from their traditional use wherein the designers have exclusive access to the FPGAs. To serve this new purpose, FPGAs are being deployed in new system architectures as offload accelerators, which used to be reserved for GPUs and ASICs, or as a bump-in-the-wire, placing them between network switches and servers, making them a first-class resource. Using Partial Reconfiguration (PR), is necessary for FPGAs in the cloud to provide OS level support including sharing, protection and portability. PR requires compile-time declaration of regions of the fabric within the parent module which should be dynamically reconfigurable, called PR regions.

Prior work uses PR by creating fixed sized slots on the FPGA allowing sharing and isolation. But this suffers from internal fragmentation, similar to the issue in managing virtual memory, due to the constraint of fitting into fixed predefined slots. [1] proposes AmorphOS for FPGAs deployed as offload accelerators, which encapsulates user modules within morphable tasks, called Morphlets. These tasks provide the necessary protection and isolation among distrustful applications, while also overcoming the fragmentation by allowing these tasks to morph and occupy the entire FPGA fabric when sharing is not necessary. Usage of module variants is a key feature which helps increase performance due to the expected performance-area tradeoff, allowing performance tuning by scaling resource utilization. When there are multiple applications, AmorphOS defaults to the slot based scheduling until it can generate a bitstream that combines the applications to occupy the entire FPGA, providing high throughput.

AmorphOS also tackles the issue of portability by providing a custom hull for different cloud FPGA systems while exposing system agnostic interfaces to the user logic. The hull provides support for connections to the FPGA peripherals and also has a software counterpart to provide libraries and system calls for scheduling and control from the host CPU.

Despite all its advantages, AmorphOS is myopic as it focuses very much on circumventing current technical limitations at the same time retaining the HDL-based FPGA programming model. In situations when sharing is necessary, it tries to improve performance by moving to a non-PR mode of operation which is not feasible in a setting more integrated (like bump-in-the-wire) than having FPGA as an offload accelerator. AmorphOS is also very restrictive in the way it approaches protection, encasing bare applications instead of opting for a higher-level virtualization of interfaces.

B. PR for Power, Energy and Area

PR, a powerful technology, is more than a mere enabler of multi-tenancy on FPGAs, but until recently has been used only for providing functionality such as with the coarse-grained role and shell approach and AmorphOS discussed in the previous subsection. [2] argues that PR can be used for much more when used to move away from the ASIC-style of static design for FPGAs and embrace a more fine-grained dynamic approach to FPGA design.

[2] applies this "PR-style" design strategy to vision applications which are used in situations where FPGA area and energy are at a premium. These applications contain modules which are not in use all the time and hence need not occupy FPGA resources when idle and can benefit from PR by only instantiating the required modules at each instant of time. Since PR time can be significant, they compare two different applications, one where the reconfiguration occurs infrequently across hours or days where PR time is insignificant, and performance comparable to the static design can be achieved. But since the dynamic design occupies less area, it fits on smaller FPGAs reducing cost and also energy in terms of lower leakage. Reconfiguration in the other application occurs between seconds to minutes in which, despite factoring in the PR time's detrimental effect on performance, energy can still be saved by the reduced resource utilization.

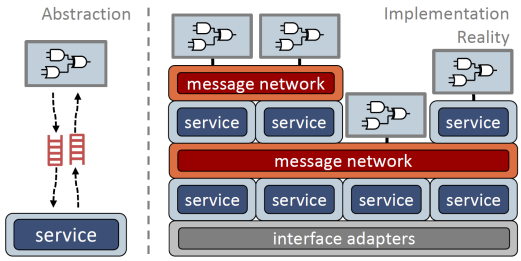

Fig. 1. Service-oriented architecture abstracts the physical connection between services, lower-level services and the memory hierarchy. Source: [3]

The paper tries to implement a fairly general purpose architecture to support PR but is only most suitable for streaming applications and is also affected by the internal fragmentation issues due to the fixed-sized slots. The approach needs to be scaled up to suit datacenter applications and have a design methodology imposed to target applications less structured than those considered in the paper.

C. Service-oriented Memory Architecture

FPGA design has been largely based on the ASIC HDL programming paradigm of describing logic. This provides immense flexibility but increases the programmers' burden to implement logic down to the most basic elements. This approach lacks the agility, reusability, maintainability and portability synonymous with software deployed on datacenters. It has also made high-performance FPGA design esoteric to experienced hardware designers.

[3] proposes a service-oriented memory architecture aimed at abstracting memory (Figure 1) using reusable modules encapsulated as services. The services expose standardized interfaces which are platform-agnostic and support communication through latency-insensitive channels. These channels can be implemented using a variety of underlying connections including point-to-point links or complex NoC topologies. The architecture enables automatic generation of larger systems using a catalog of modules developed by domain experts.

The framework is evaluated using a BFS design implemented by decomposing memory and data-structure operations from the application code to form services connected using point-to-point channels. The abstraction adds minimal overhead in performance and resource utilization to an optimized FPGA implementation of BFS. It also eases the creation of multiple versions of the design using variants of service modules to access different underlying memory or analyze and provide performance enhancements.

This architecture can be extended beyond memory support, to offer static composability by restricting the design process to creation and composition of services as opposed to the ASIC programming paradigm. The addition of PR can usher a dynamic aspect to this composition of services and promote more expressive virtualization of FPGA and server resources.

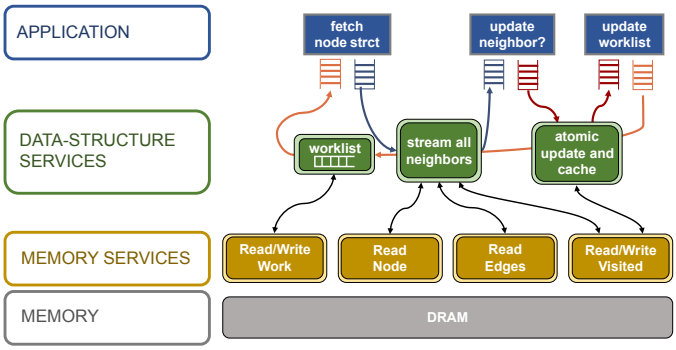

Fig. 2. Composability across the stack in the service-oriented architecture can be exploited during runtime to provide different functionality or tradeoffs

III. Harnessing Tradeoffs at Runtime

Tackling the challenges of multi-tenancy, agility and portability for FPGAs deployed at datacenter scale using PR, demands the development of a new future-proof framework for FPGA design. Compute problems targeting FPGAs benefit from having specialization to extract data-parallelism unavailable in other general purpose settings. Such a framework must be general enough to map these applications at low overhead, while at the same time conferring specialization through the use of PR. AmorphOS [1] supports this functionality but in harnessing performance, breaks the abstraction. This work aims to create a framework for dynamic composability that combines the OS abstractions proposed in AmorphOS with PR-style design [2] in a service-oriented design paradigm [3].

The envisioned framework rethinks the programming paradigm to prioritize making design decisions and tradeoffs at runtime, i.e., on an input-to-input and FPGA load basis than under design-time constraints. The design is composed of services, communicating through latency-insensitive channels, equipped with tuning knobs in the form of parametrization, or catalog of implementations, which control various metrics characteristic of the service. Using these knobs, the module can be dynamically altered as a function of runtime constraints provided by applications co-scheduled on the FPGA, the FPGA system itself and/or the specific input to the application. This ability enables the framework to harness design tradeoffs at runtime in contrast to static ASIC design style.

To illustrate the advantages and challenges of implementing these principles, the BFS design from [3] is used for a case-study to apply them to the family of algorithms involving graph traversals. The dynamism helps the design achieve lower costs on certain inputs while retaining performance.

IV. Dynamic Composability in Graph Traversal

Figure 2 shows a service-oriented abstraction of graph traversal-based applications. The decomposition of the design into services naturally falls into a hierarchy when keeping memory abstraction in mind. These family of algorithms can take advantage of composability across the stack: applications such as BFS and SSSP use the same set of graph traversal services; different graph representations such as compressed sparse row (CSR) or a dense bit-mapped adjacency matrix can be traversed by modifying a subset of the data-structure services; caches can be introduced into the memory services; and at the lowest end, memory technologies could be swapped, all to improve performance when possible and beneficial or change functionality when required.

A. Implementation

The BFS application was studied at the data-structure and memory services levels by porting the service-oriented implementation to a server-grade FPGA board. A generic shell was built in order to add a first-level abstraction to the peripherals to make porting to the FPGA easier. The shell consists of IPs to interface with the FPGA DRAM, support PCIe memory-mapped IO and DMA, and control the PR flow. It also includes switches to connect all the IPs together and ensure proper clock crossings across the IPs' domain. Special care was taken to pipeline the switch connecting to the DRAM to ensure timing and full bandwidth availability. Though not used currently, the rest of the design is floorplanned into a partition to mimic the role and shell coarse-grained PR approach, and hierarchical PR is used to reconfigure modules within the role.

Composability was studied at two levels in the hierarchy: (i) A direct-mapped, write-through cache was introduced in the visited array's read/write service to study the tradeoffs between performance and size of the cache which affects the applications energy and resource utilization and (ii) Graph traversal services were implemented for CSR and dense matrix based graph representations to further study the interaction between performance and memory utilization. The dense graph uses a bit matrix to represent the graph's adjacency matrix while the dense traversal service uses priority encoders to improve performance even in sparse graphs.

Challenges: Adding PR support to the services framework underscores limitations in current FPGA implementations. Floorplanning service modules into PR regions, leads to wasted resources due to the homogeneity across resources in the fabric. For example, since SRAM blocks are spread out across the fabric, a region that encompasses enough cells to fit the cache, invariably includes more than enough logic resources. Moreover, due to the lack of a general network topology, timing across PR boundaries needs to handled delicately. The latency-insensitive interfaces allow for a provisional solution of adding FIFOs to decouple timing. Also, dynamic swapping of modules requires the ability to start and stop a service while handling its context safely. The write-through cache supports non-preemptive context switching since its internal state is limited to only pending transactions.

V. Evaluation

The results in this section are obtained on a Bittware 520N-MX (Stratix 10 MX2100) FPGA using single channel DDR4 memory. The PR bitstreams and graph data reside in the on-board DRAM and the time taken to load them from the host is not included in the measurements. PR time is factored out assuming infrequent use and better support in future FPGAs.

TABLE I
BFS BENCHMARK GRAPHS AND SIZES

Graph	Nodes	Edges
rome99	3.3k	8.8k
cond-mat	40k	351k
USA_FLA	1.1M	2.7M
rmat_256k	256k	4.2M
rmat_1m	1M	16M

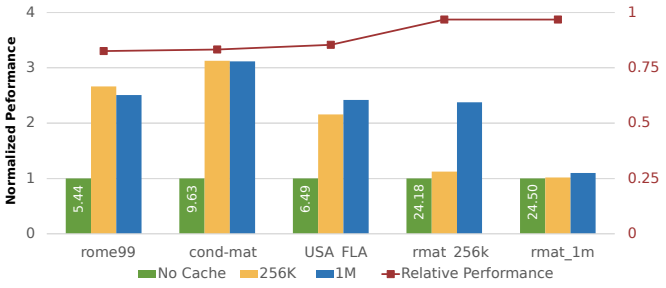

Fig. 3. Bar Chart of Performance of the service-oriented BFS implementation, measured in million traversed edges per second (MTEPS), normalized to the base case of no cache showing graph dependent variability and Line Chart of relative performance of the PR version (average over cache sizes) with respect to a static implementation showing minimal loss

A. Caching

The benefits of variable caching were evaluated using standard and synthetic graph benchmarks (Table I) as inputs. Performance in three different caching scenarios are compared in Figure 3. The design without a cache allows the upstream service to directly access memory to avoid the extra latency of FIFOs in the PR region path which were introduced to meet timing. Such alternate paths would be naturally available with a more generalized framework to connect services.

While none of the graphs lose from having the largest cache, some graphs can reach their peak performance with a smaller cache (`rome99` and `cond-mat`), while larger graphs like `rmat_1m` don't benefit from having even the largest available cache. In a situation when multi-tenant FPGA fabric is charged based on resource utilization as well as time, this trend could be exploited to reduce cost by opting for smaller caches or removing it altogether without hurting performance, allowing demanding applications to use more of the fabric. Reduced area also translates to lower static and dynamic power consumption even when not sharing the FPGA. Adding PR leads to a drop in performance due to the increased FIFO latency when accessing the DRAM. Smaller graphs are affected to a greater extent, but on average the overhead is minimal compared to the benefit of having the cache.

B. Graph Representations

To evaluate the dense and sparse graph representations, synthetic graphs with 16k nodes with varying sparsity were generated using the Graph500 generator with default values and node degrees following the power law. Figure 4 plots the ratio of performance (in MTEPS) and amount of memory used by the sparse vs dense graph representations. Greater than one

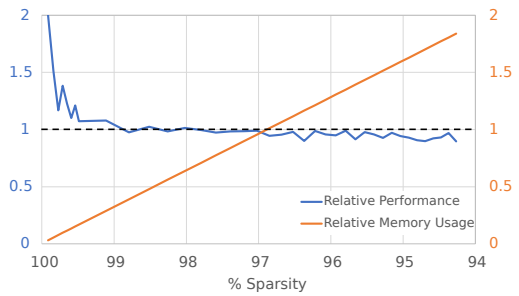

Fig. 4. Performance and memory utilization of the sparse CSR relative to the dense implementation as a function of adjacency matrix sparsity shows that at lower sparsity, CSR performs worse while using more memory

relative performance is favourable for the sparse representation while greater than one relative memory usage is favourable for the dense representation since less memory usage is better.

As expected the sparse representation performs much better in both metrics at high sparsity. The dense representation performs at least as good as the sparse representations starting at a sparsity of around 98%, and after 97% the dense representation is also economical in memory usage. This shows that dense representations can achieve better performance while saving memory depending on the sparsity of the input graph and with dynamic composability, the implementations can be picked at runtime based on the input to get the best performance. It can also be used to provide the necessary functionality in cases when the conversion between input representations is infeasible compared to swapping hardware modules.

VI. CONCLUSION

FPGAs are entering a new domain as devices central to specialized compute and this demands innovative approaches to break from the prior regime of operation. The work proposed a framework to make FPGAs more reactive and responsive to changing external and internal environments, key to survival in the server landscape. Using the service-oriented architecture to guide hardware design and PR to take a dynamic approach to module composition, the framework is able to make intelligent tradeoffs by considering static as well as runtime factors. The case-study on the BFS algorithm shows clear benefits in making input-dependent design decisions enabled by this framework. The proof-of-concept can be extended to build a more general and reusable architecture, which can be used to inform directions in developing novel FPGA architectures.

REFERENCES

[1] 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.

[2] 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)*, (Barcelona, Spain), pp. 129–135, 2019.

[3] J. Melber and J. C. Hoe, "A service-oriented memory architecture for fpga computing," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*, (Gothenburg, Sweden), pp. 91–97, 2020.