aFabric: Towards a Holistic View for Managing Hardware Accelerators in the Cloud

Changhao Wu
Brown University
Providence, RI, the United States

Theophilus A. Benson
Brown University
Providence, RI, the United States

ABSTRACT

As hardware accelerators are widely used in the cloud to reduce response time, it is impending to orchestrate and unify heterogeneous accelerators. In this paper, we describe an operating system paradigm for heterogeneous accelerators in the cloud and introduce our sketch design of an operating system, aFabric, using P4 language as the programming language for offloaded accelerator programs. Finally, we elaborate on the evaluation plan as the guideline of concrete design and implementation of aFabric in the future.

CCS CONCEPTS

• **Computer systems organization** → **Real-time operating systems**; • **Networks** → *Cloud computing*; *Programmable networks.*

KEYWORDS

Operating system, Heterogeneous hardware, Cloud accelerators

ACM Reference Format:
Changhao Wu and Theophilus A. Benson. 2020. aFabric: Towards a Holistic View for Managing Hardware Accelerators in the Cloud. In *Student Workshop (CoNEXT'20), December 1, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3426746.3434057

1 INTRODUCTION AND MOTIVATION

With the end of Moore's law and the lure of dedicated hardware's high performance and energy-efficiency, cloud providers are exploring a wide range of accelerators, *e.g.*, GPUs, FPGAs, and ASICs. As they provide attractively significant performance and efficiency gain, such as throughput improvement, low energy cost, and latency reduction, providers use them to improve application performance, increase revenue and overcome CPU performance bottleneck.

However, using accelerators for the applications are pain-staking and error-prone, since developers have to deal with different architectures, and hardware have different strict resource constraints, which are things that programmers pay less attention to during development. In addition, modern cloud providers are still providing individual accelerators as computing instances [1], meaning that cloud providers leave the hardware management tasks to users, and users will waste the part of computing power they never use.

CoNEXT'20, December 1, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-8183-3/20/12…$15.00
https://doi.org/10.1145/3426746.3434057

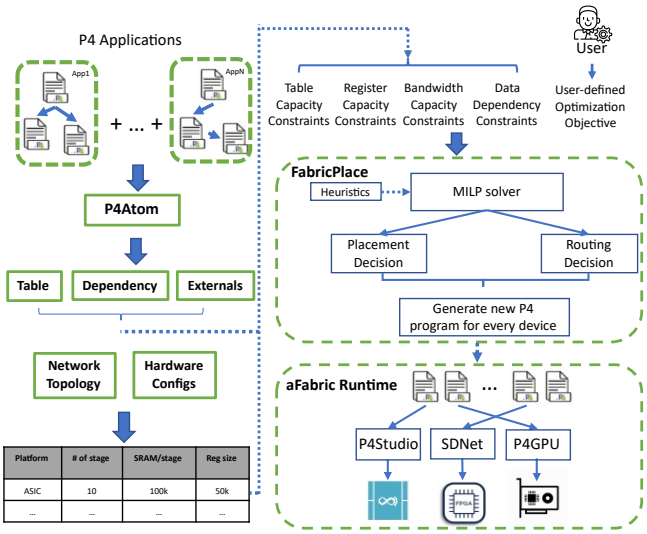

Figure 1: aFabric's workflow

Therefore, it is desirable for cloud providers to have an operating system that manages heterogeneous hardware and programs. First, each heterogeneous target offers a different trade-off in design space, *e.g.*, ASICs with high throughput but limited memory size. A global scheduler can make trade-offs and map programs to the best targets available. It can be even better, if a program can be automatically broken down into pieces, meaning that the allocation algorithm has more flexibility planning resource usage; and programs which exceed the hardware resource constraints are able to be executed. Second, such an operating system makes the underlying heterogeneous architectures transparent to programmers, since all arduous management tasks, such as data isolation and device management, are on the operating system. Third, it will become beneficial to cloud as well, since cloud will understand users' intentions, which is an opportunity for an operating system to achieve global objectives, *e.g.*, minimizing cloud's energy cost.

To build such a system, we are facing three problems. The foremost problem is scheduling different users' programs upon accelerators. Since accelerators are not usually x86 architecture, which makes the scheduling technique applied on modern OSs not suitable. In addition, most accelerators do not have supports for program co-existing, which makes resource multiplexing difficult to achieve. Though ViTAL [3] built a system for virtualizing FPGA programs, it is hard to apply it to heterogeneous hardware. Second, as different accelerators have different specialties, *e.g.*, GPUs for large memory capacity but relatively high latency and ASICs for low latency but limited size of memory capacity, a sophisticated scheduler is required to allocate different resource for different types of program demands, while effectively improve the hardware's utilization rate

and reduce the hardware cost. Third, a important part in an OS is memory isolation, which does not has the built-in support from accelerators; and some applications are executed in a distributed fashion, this further raises the bar of memory isolation.

We propose aFabric, an OS for heterogeneous accelerators. We treat accelerator programs as control flow graphs and effectively decompose them into pieces. Instead of scheduling programs temporally, we schedule all accelerator programs by space. In this way, scheduling becomes a program placement problem; and aFabric will properly encode different resource and performance constraints that aFabric extracts from the program pieces, and receives from user specific intentions. In addition, aFabric encodes the information about the hardware configurations and accelerator topology. Finally, the placement problem becomes an optimization problem; and we design heuristics to solve the problem and address the scheduler's scalability problem. This optimization process places program pieces with different demands to appropriate locations to achieve global optimality. Fine-grained program placement also improves the hardware utilization rate. After the placement decision has been made, aFabric will generate new programs by merging program pieces for every piece of hardware and inject code to ensure memory isolation.

In this paper, We recognize the benefits and challenges to build a heterogeneous accelerators OS and present a heterogeneous accelerators OS paradigm receiving programs written in P4 language as the programming language for accelerators.

2 DESIGN OF AFABRIC

As shown in figure 1, aFabric receive programs defining the overloaded computing tasks and dataflow specifications from users and analyze and decompose P4 programs into P4 tables and schedule them based on hardware constraints information from the cloud and global management objectives. As we previously mentioned, the scheduling program will become a program placement problem. Since we need a hardware abstraction across heterogeneous hardware to place P4 programs, we choose the PISA architecture [2] abstraction. Although this abstraction places some strict constraints of hardware functionality, it is easier to extract the resource constraints and suitable for placing P4 programs, which use the language that follows the PISA architecture. In addition, to make the accelerators fabric transparent to users, aFabric allows users provide configurations describing dataflow between programs; and aFabric takes the dataflow constraints into consideration during program placement, ensures data isolation, and maintain correct data exchange between programs. If programs have to migrate to different locations due to change of users intentions or satisfaction of the global objective, aFabric will migrate programs' state correctly.

P4Atom takes in P4 programs, and analyzes and extracts resource requirements and data dependency from the programs. To preserve the semantic correctness, P4Atom finds the data dependency between tables. Since a P4 program may span over multiple switches, data dependency across devices should also be preserved. In addition, a P4 program may contain registers serving as stateful storage. Therefore, P4Atom will also inspect the P4 program and find out a possible solution, *e.g.*, piggybacking data or sending extra packets to update, to maintain the data consistency.

FabricPlace is fed with the P4 tables from P4Atom and hardware information and configurations provided by the cloud, and generates a model that covers resource constraints and data dependency. This model finally becomes an optimization problem, *i.e.*, how to place tables on given hardware resource while preserving constraints and program correctness, and respecting certain objectives. This optimization can be achieved by using a Mixed-Integer Linear Programming(MILP) solver. After the decision is made, FabricPlace generates the P4 programs to be placed on hardware.

aFabric Runtime serves as the operating interface for users. Since users join, leave, and update programs and configurations all the time, a runtime broker is needed. aFabric Runtime's role falls into 3 parts. 1) *Schedules programs.* Users submit or revoke their tasks to aFabric runtime. Then it will invoke FabricPlace to generate an up-to-date placement decision, which takes resource defragmentation into consideration. 2) *State migration.* aFabric runtime will orchestrate the hardware to migrate the stateful information for one device to another. 3) *Ensure data isolation.* P4 programs are accompanied by P4 control plane applications, which are granted write/read permission to change or read data plane's tables or registers. All write/read operations from control plane are submitted to aFabric Runtime, which will arbitrate whether the operations are permitted or not and forward the permitted operations to the correct data plane devices.

3 FUTURE EVALUATION

The evluation of aFabric prototype will be separated into 3 parts.

Correctness: 1) The correctness of programs execution. Although a program may span on multiple device, the functionality should be equivalent to the original one. 2) The correctness of privacy maintenance. There should be tests to check if it is possible to write/read other users data from both data plane and control plane.

Scalability: 1) it is known that MILP is an NP-hard problem, so as the number of devices increases, a MILP solver will not generate a placement decision in time. Therefore, heuristics will be deployed. An experiment should demonstrate how quickly a placement decision can be made as topology grows. 2) Hardware compilers will take a significant long time to compile the code. Therefore, even after the aFabric has made the decision, it will take extra time to compile. How frequent the programs can be scheduled should be evaluated to show that it is feasible to host many users' intentions.

Optimality: 1) We should demonstrate the optimality of our resource allocation heuristics and how it is better than placing individual programs on individual devices. 2) We need to prove that hosting programs on heterogeneous hardware helps cloud providers satisfy different users' demand, *e.g.*, high memory demand.

Acknowledgments We thank the anonymous reviewers for their invaluable feedback. This work was supported in part by NSF grant CNS-1749785.

REFERENCES

[1] AWS. Amazon ec2 f1 instances. https://tinyurl.com/y6cp3zeq, 2020.
[2] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. SIGCOMM '13, page 99–110. ACM, 2013.
[3] Y. Zha and J. Li. Virtualizing fpgas in the cloud. In *ASPLOS 2020*, pages 845–858.