Research Statement

1 Introduction

As the gap between the demand and availability of compute performance & efficiency of modern processors widens, hardware specialization is increasingly being seen as a viable solution. In the last several years, areas such as deep learning, scientific computing, and other fields have seen a large number of ASICs being proposed — both in academia and industry. While specialized hardware accelerators bring a great deal of performance and efficiency improvements to the table, they pay a large cost by losing the flexibility and programmability that general purpose processors offer. A significant number of hardware accelerators proposed in the community are often very specialized to a specific algorithm or worse; one implementation of an algorithm.

This inflexibility poses two major challenges: **(1) Across algorithms:** As algorithms evolve in fast-changing fields, the effort needed to design new ASICs accordingly is very high and is not sustainable. Additionally, given the sheer number of algorithms deployed in modern computing, it is unfeasible to design and integrate ASICs for each of them. **(2) In an algorithm:** As the problem scale varies based on use cases or other reasons, hardware accelerators display sub-optimal efficiency due to their rigid design in terms of data locality and access patterns [1].

In this light, as the volume of new ASICs designed continues to rise, tackling the open research problem of developing specialized hardware that provides high efficiency and performance without fully sacrificing flexibility and programmability is the need of the day.

2 Research Overview

Motivated by the above challenges, my research towards specialized hardware design has three main thrusts: **(1) Intra-algorithm flexibility**, **(2) Inter-algorithm flexibility**, and **(3) Reusable hardware abstractions**. Broadly, my aim is to endow flexibility and programmability to specialized hardware accelerators in different degrees without losing efficiency compared to a similarly provisioned contemporary inflexible solutions.

2.1 Intra-algorithm Flexibility

The major goal in this thread of research is to codesign the hardware accelerator and the associated software to enable the user to program the targeted algorithm in different flavors to gain the best possible efficiency. For example, an accelerator designed for 2D-Convolution on images can be significantly benefited if the user can perform loop interchange [3] based on the image dimensions. Developed flexible accelerators enable users to tune their target algorithm on the hardware — hence it is essential to support a framework to search for efficient parametrization of the problem. Essentially, this turns into an optimization problem whose heuristic can be power efficiency, data reuse, utilization etc., as described in Equation 1.

$$runtime\_parameters = \mathbf{f_{opt}}(hardware\_parameters,\ problem\_parameters) \tag{1}$$

In our recent work [1], we developed a highly flexible accelerator for video understanding that proposed an accelerator flexibility in terms of tiling, loop order and parallelism that improved perf/watt by 5.1× over the inflexible baseline. An interesting observation this work made is that, endowing flexibility improves the performance of an accelerator compared to an inflexible counterpart, which points to that fact that *intra-application flexibility* is a sweet spot that architects should exploit. This work helped me receive a PhD fellowship from Facebook to be developed further.

2.2 Inter-algorithm Flexibility

Most of the contemporary accelerators are at the granularity of applications (App) as depicted in Figure 1, which limits them from being used across different algorithms from the same domain. It is unfeasible to continue to develop more accelerators at the App layer due to the associated design, verification, and integration costs. This motivates us to develop accelerators that can serve an entire domain of compute workloads. General purpose computers make this possible by working at operation (OP) granularity, but this level of fine-granularity not only loses efficiency, but also offers lower performance.

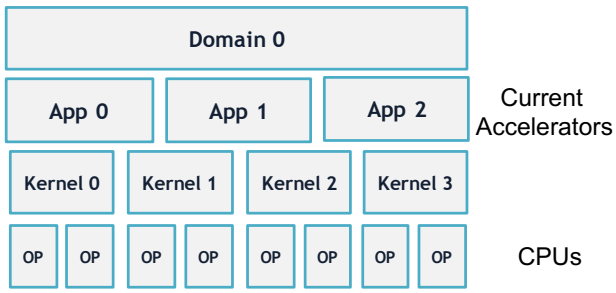

Figure 1: Different abstraction levels for hardware design.

A key observation here is that, several important compute domains constitute of a finite number of compute primitives (Kernels) that are used across applications. Therefore, developing efficient hardware blocks to accelerate these primitives and tightly integrating them with an efficient task scheduler potentially can bring large benefits across a stack of applications. In this thrust, I focus on developing such Kernel level hardware accelerators, hardware schedulers to support them and compilers to generate dataflow graphs to be executed on. As a proof of concept, we designed an out-of-order speculative hardware task scheduler for a system consisting of several Kernel level accelerators for DSP, and demonstrated up to 12× performance improvement over the baseline (unpublished).

2.3 Reusable Hardware Abstractions

A major issue with specialized hardware accelerator design is the time and cost associated with the silicon design and verification. However, I believe that there exists a large opportunity to alleviate this, as a significant portion of these accelerators is reusable across multiple designs. Such reusability exists for memory hierarchy design, address generation state machines, processing elements, and NoCs to name a few. Our recent work developed a reusable, composable hardware abstraction for on-chip buffers called *buffets* that demonstrated 1.53× EDP advantage over scratchpads.

Our current work explores a programmable pattern generator aimed at efficient address generation for sparse-dense data structure access. This is a key component in developing highly programmable accelerators, as address generation holds the key programmability. We are seeing promising results on several important sparse-dense workloads in tensor algebra across a whole range of real-life applications.

References

[1] Kartik Hegde, Rohit Agrawal, Yulun Yao and Christopher W Fletcher. *Morph: Flexible Acceleration for 3D CNN-Based Video Understanding*. 2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO).

[2] Michael Pellauer, Yakun Sophia Shao, Jason Clemons, Neal Crago, Kartik Hegde, Rangarajan Venkatesan, Stephen W. Keckler, Christopher W Fletcher and Joel Emer. *Buffets: An Efficient and Composable Storage Idiom for Explicit Decoupled Data Orchestration*. 2019 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).

[3] Wolf, Michael E and Lam, Monica S. *A data locality optimizing algorithm*. ACM Sigplan Notices 1991.