Motivation: Affordable, Sustainable, and Accessible Hardware Customization

Moore's Law can't continue forever. We have another 10 to 20 years before we reach a fundamental limit.
— Gordon Moore (2005)

Though the exponential computer performance growth in the past decades drives the flourishing of the computer engineering, it also imposes significant unsustainability on both hardware vendors and customers. To keep dominating the market while preventing potential competitors, several extremely large hardware vendors monopolize hardware design innovations by having large teams of software/hardware engineers to iterate over their products at each generation. Consumers have to deprecate their devices yearly (e.g. iPhone and Nvidia GPU) for the performance improvement.

40 years after Moore's Law, Gordon Moore again predicts its demise and advocates fundamental changes to the current hardware design paradigm. Many prior works have proven the promise of *domain-customized* hardware (a.k.a. accelerators) by achieving orders-of-magnitude speedup and energy saving. Though promising, they also bring new challenges to both system and application developers: Due to the infancy of accelerator software stack development, accelerator programmers still rely on low-level and error-prone interfaces, and the excessive cost of building a new computing system can hardly be justified to have an accelerator for each application domain of interest, which makes it even harder for small businesses to bring their hardware innovations to the market.

My ultimate research goal is to build a sustainable, accessible, and inclusive design paradigm to unify the hardware/software co-designed ecosystem, which not only prolongs the chip lifetime, but also eases the difficulties of all the aspects of hardware customization. Thus, my research interests span designing and analyzing specialized hardware mechanisms, developing compilation techniques and abstractions for heterogeneous hardware, as well as automating software/hardware co-design.

Accomplished Works: Automating Customized Hardware Design

Towards my ultimate research goal, my doctoral studies answer three fundamental questions for building such a framework that diversifies, automates, and democratizes *customized* hardware design: 1. How to formalize an accelerator design space? 2. How to unify the compilation for this design space? 3. How to effectively search this design space? Besides these three questions, I also apply all the insights and principles I learned from this process to applicable domains.

Universal Accelerator Design Space Accelerator designs used to be very adhoc: those developed specialized mechanisms can hardly be reused in other applicable domains. Therefore, it is critical to define a formalized architecture design paradigm and space. Ideally, each innovative mechanism can be included as primitives in this space, and accelerators can be designed and implemented by composing these primitives.

My insight is that the *spatial* architecture paradigm is especially promising in terms of performance, flexibility, and scalability. *Spatial* architectures typically refer to the architectures that expose their computing resources and on-chip network to the software/hardware interfaces. These architectures can be composed by a set of primitives, including computing resources, memories, controllers, etc., connected by on-chip networks composed by switches. Scalability can be achieved by connecting arbitrary resources. Hardware innovations can be done by extending the primitives, and accelerator composition can be done by connecting these primitives — versatile hardware organization can be approximated, including SIMD vectorization, systolic arrays, and application-specific datapaths to achieve comparable (or even better) performance.

My prior works [7, 1] have proven the promise of this rich design space. For example, because of the speedup of 5G, digital signal processing (DSP) algorithms require orders-of-magnitude acceleration. These algorithms are mostly inductive matrix factorization, which were considered hard for conventional ad-hoc spatial architectures (e.g. systolic arrays). However, by flexibly composing computing resources with different execution model, and extending the memory access unit, I built a next-generation DSP accelerator, that outperforms a high-end server CPU by 1000× perf/area, and achieves comparable performance compared with equal-provisioned ASIC collections with only half the area. The area saving comes from the unification of the computing resources achieved by reconfigurability.

Another example is the sparse processing unit (SPU) [1]. My collaborators and I notice two highly data-dependent program behaviors, key-value join and alias-free indirect memory access, widely exist in databases, sparse linear algebra, and graph processing. We together built a back-pressure based execution on the spatial datapath to accelerate the key-value join, and extended banked scratch memory for parallel indirect memory addressing.

To sum up, this spatial composition methodology enables a well-formalized design space for spatial architecture, and all the innovations integrated into this design space can easily be reused for further design automation.

Universal Accelerator Compiler Because of the infancy of the compilation techniques for the aggressively reformed execution model and ISA, it requires intensive human efforts to recognize the program behaviors of interests, manually transform and map them to specialized hardware by programming in error-prone low-level interfaces. It is even more challenging to build a compiler under the context of design automation. Instead of targeting a specific architecture, the compiler should be able to target all the design points within this space.

My insight is to construct the compiler in a modular way, where hardware-feature specific transformation passes can be composed according to hardware capabilities. For each compilation that makes use of an optional hardware feature, there will always be a fallback transformation that escapes this feature for a successful compilation when unavailable. By applying this insight, I present my compiler works [6, 4] that target a high-level language, C, to specialized accelerators.

To keep compatibility with a legacy code base, and find a balance between compilation works and user experience, my compiler still relies on some additional information on array aliasing and instruction concurrency from application developers by annotating the programs with several modest pragmas. By taking advantage of these additional information, the compiler detects the program behaviors of interests and map them to the hardware by adopting a feedback-directed resource provision. To explain, the compiler first tries to apply all the transformations that are available on the underlying hardware. If fails, it iteratively disables the transformation with the least estimated performance impact until succeeds. My compiler is able to target a wide range of application domains, including MachSuite, DSP, and dense linear algebra kernels, achieving 2.3× speedup over high-end server CPUs, while retaining robust across arbitrary combinations of the hardware features.

Beyond the scope of the developed compilation techniques, I argue that, potentially one day, there will be hundreds of programmable accelerators, but it will never be practical to build thousands of software stacks for them. Therefore, the promise of a unified software stack is to push the accessibility of programmable accelerators. Moreover, software behaviors of interests naturally encode the design demands of an accelerator, and a compiler is the aspect that is best aware of these encoded opportunities. This awareness should be well utilized for design automation.

Automating Specialized Accelerator Design Domain-specific accelerators are extremely costly to design, taking teams of hardware and software/compiler engineers many months. Therefore, it will be highly desirable to have an automated approach for the accelerator design. Ideally, for a given set of applications of interests fed to a design-space exploration (DSE) framework, this automated approach should yield an accelerator deeply specialized.

To achieve this "ideal approach", my work, DSAGEN [5], breaks this down to three steps. First, a well-formalized design paradigm and design space for spatial accelerator is defined. Each design point can be represented by composing a set of hardware components, including computing resources, memories, controllers, switches, etc. Second, the design requirements of the accelerator are extracted from the target program domain. These requirements are naturally encoded in program behaviors of interests and can be detected and extracted by the compiler. Lastly, a design space explorer iteratively evolves the candidate hardware, from an initial design, guided by estimating the hardware/software affinity. Because of the frequent modification to the hardware, it is impractical to neither re-run all the software for the performance nor re-synthesize the hardware for the cost. Therefore, we build an analytical models to estimate the software performance by estimating the instruction-level parallelism on the spatial architecture, and a regression model to estimate the hardware cost. Then we use simulated annealing to iteratively modify the candidate hardware until the objective function, pref/area, converges.

This work significantly eases the difficulties of building a new full-stack reconfigurable accelerator. An accelerator can be done within a single day what would have previously taken our research group months to complete, and it also saved 30% area over the manual design. In addition, the ultimate goal of this work is to finally unify and democratize the specialized accelerator research. Therefore, I held two tutorials in architecture top-tier conferences (MICRO 2020 and MICRO 2022) to promote our framework. Though the current implementation is a proof of concept, the infrastructures are open source[1]. It has attracted many early users all across the world including Illinois, Michigan, KAUST, ICT, ISCAS, UCR, and Fudan.

Besides easing the accelerator design process, this approach can also be applied to improve the usability of FPGAs.

Revolutionizing FPGA Programming FPGA programming has been a persistent challenge for decades. Programming FPGAs in register-transfer language is error-prone and tedious, considering the massive concurrency and timing constraints. An alternative is high-level synthesis (HLS). Application-specific accelerators can be generated from a high-level language with several additional hints. Though it enables more productivity, the lengthy process of generating a fixed-function accelerator will be repeated, when shifting the target application.

My approach [2] revolutionizes the FPGA programming paradigm, by applying my accelerator design automation approach. The generated programmable accelerator can be deployed as an overlay on the FPGA to achieves shorter hardware generation time, faster reconfiguration time, and more programmer-friendliness, while retaining comparable performance over HLS. By generating a specialized accelerator for the given set of applications together, common efforts like synthesis and FPGA physical design and can be unified to save compilation time. FPGAs take an order of milliseconds to load the hardware configuration, while my generated programmable accelerator takes less than one microsecond, which is orders-of-magnitude saving on reconfiguration. Lastly, my high-level programming interface is also more developer friendly: for example, Xilinx HLS has hundreds of pragmas, but I only have 2 pragma extensions; HLS requires awkward coding style (e.g. loop perfection) while our compiler is more robust due to modular optimizations and performance predictions.

To the best of my knowledge, this work is the first work that reproposes a promising FPGA programming paradigm. This approach can potentially be a challenger or even replacement for the existing HLS flow because of all its advantages. This work is awarded as Best Paper Runner-up at 55th MICRO.

Unleashing the Power of Tensor Domain-Specific Language Tensor domain-specific languages (DSL) are known to be productive and powerful tools to write portable tensor programs for image processing and deep neural networks (DNN). By taking advantage of the domain knowledge encoded in DSL, my prior works push the compilation techniques for heterogeneous hardware even further.

Because of the increasing demand of computation on the DNN models, hardware vendors, like Intel and NVIDIA, have provided specialized instructions for them: Horizontal reduction accompanied with mixed precision among input and output operands makes the compilation challenging for general-purpose programming. The applicability of these extensions is hindered by the lagging of compilation techniques: Developers could only manually write assembly code, or call vendor-provided kernel libraries. To address this issue, my work [3] presents a framework that automatically detects the applicability of instructions with these semantics and rewrite the program. This work is the *first* work that takes reduction and mixed precision into consideration for auto-vectorization, and I named this technique *tensorization*. This technique is also adopted by further research from UCB and Peking University.

The principles of spatial accelerators can be extended for promising new technologies. My ongoing work incorporates tensor DSL, and spatial architecture to a full-stack scalable processing-in-memory (PIM) system. PIM is a promising hardware paradigm, which brings computation nearer to data for higher data bandwidth and energy saving, and achieves extremely high computation density at low cost over traditional vectorization. Though promising, PIM architectures have even worse software-stack support than accelerators. In this work, we show that our tensor-DSL-stack can dramatically simplify the programming interface of a PIM system by mapping the program parallelism to the hierarchical parallelism in a spatial-PIM system.

Future Works: Make Accelerators Everywhere

Though my prior works already present a way that significantly reduced the effort of accelerator design, broad adoption of accelerators are still facing significant challenges on the design methodology, system-level abstraction, as well as datacenter level applicability.

Pushing the Applicability of Design Automation Specialized accelerators have been widely adopted, including data centers (e.g. Google TPU, and Microsoft Brainwave), mobile devices (e.g. Apple & QualComm SoC), and real-time

[1]Available at `https://github.com/polyarch/dsa-framework`

systems (e.g. Meta Oculus, and Tesla Autopilot). All these accelerators all have different first-class design concerns. For example, in data centers, they are the energy efficiency and the throughput of data processing. In mobile devices, the area and the thermal can have the highest priority. For real-time systems, like self-driving or AR/VR system, the latency requirement should first be fulfilled before any other concerns. Moreover, for applications with mostly serialized behaviors, the single-core performance should be well optimized, including the ILP and clock frequency. To fulfill all these different design concerns, broad research questions are open: new software/hardware co-design mechanisms should be proposed to enrich the design space; to guide the DSE to converge to the prioritized constraint, objective functions that encode different requirements should be carefully considered; moreover, to accurately estimate the constraints, like thermal and frequency, new models should be developed; considering the synergies across all these aspects, the design space may exponentially grow, so all search strategies should also be improved.

I am currently actively seeking solutions to all these questions above for next-generation accelerator design automation.

Virtualizing Spatial Accelerators The conventional system-on-chip (SoC) design paradigm can be described as the sea-of-ASICs approach, which is widely adopted by all major mobile SoC vendors, including Apple, QualComm, and Motorola. Though effective, it is especially expensive and time-consuming to redesign and integrate the accelerators on the SoC for every generation, considering the increasing number of specialized blocks on chips in the past decades. An alternative vision is that a multicore-reconfigurable accelerator can be used to unify and replace all the SoC accelerators. However, unlike ASICs — all the resources are dedicated to the application for which it is designed — a reconfigurable accelerator can be shared across multiple concurrent applications, but the resource management strategies were not well studied yet. I believe virtualization is a promising technique to solve this issue — specialized virtualization techniques should be developed for spatial architectures. First, prior operating systems either manages programmable devices, like GPU and TPU, that are loosely coupled with the host core, or manages application-specific blocks tightly coupled with the host core, like Apple SoC. In this case, an OS support for programmable accelerators tightly coupled with the host should be developed. Second, context switching/isolation on a thread-based execution is already well studied. Because all the data, including text segment and register files, are centralized, context can easily be isolated and switched. However, the spatial architectures adopt a fully distributed instruction execution, which brings challenges to porting existing techniques to this paradigm. All these challenges open "blue sky" research questions on OS-accelerator co-designs.

Self-Evolving Overlay System in Data Centers I envision that once the software ecosystem, including but not limited to compiler, OS, and design automation tools, for specialized accelerators is getting mature, their applicability will be pushed even further. The applicability of accelerators will no longer be just limited to ML models and DB queries. I define this new research area as adopting re-synthesizable accelerator overlays in data centers. This opens unique questions to this emerging architecture paradigm. Because of the heterogeneity of the synthesized accelerators, it is challenging for the system to schedule the tasks while retaining the tradeoffs among performance, load balancing, and binary sizes. It is highly desirable to take advantage of the flexibility enabled by re-synthesis to fulfill different real-time execution frequency and QoS requirement by dynamically resynthesizing and allocating accelerators. Ideally, all the legacy applications should be automatically profiled and ported to accelerators. When it comes to newly developed applications, the system should decide if a re-synthesis is required or just compiled to an existing overlay. If so, the subset of applications for synthesis should also be automatically selected. Lastly, because the data center architecture is aggressively reformed, it is also critical to maintain the compatibility with external data centers.

If I were hired as a faculty member, I would like to educate my students how research is conducted by exploring all these questions discussed above with them.

References

[1] Vidushi Dadu, **Jian Weng**, Sihao Liu, and Tony Nowatzki. "Towards General Purpose Acceleration by Exploiting Common Data-Dependence Forms". In: *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO '52. Columbus, OH, USA: Association for Computing Machinery, 2019, pp. 924–939. ISBN: 9781450369381. DOI: 10.1145/3352460.3358276. URL: https://doi.org/10.1145/3352460.3358276.

[2] Sihao Liu=, **Jian Weng=**, Dylan Kupsh, Atefeh Sohrabizadeh, Zhengrong Wang, Licheng Guo, Jiuyang Liu, Maxim Zhulin, Rishabh Mani, Lucheng Zhang, Jason Cong, and Tony Nowatzki. "OverGen: Improving FPGA Usability through Domain-specific Overlay Generation". In: *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2022, pp. 35–56. DOI: 10.1109/MICRO56248.2022.00018.

[3] **Jian Weng**, Animesh Jain, Jie Wang, Leyuan Wang, Yida Wang, and Tony Nowatzki. "UNIT: Unifying Tensorized Instruction Compilation". In: *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2021, pp. 77–89. DOI: 10.1109/CGO51591.2021.9370330.

[4] **Jian Weng**, Sihao Liu, Vidushi Dadu, and Tony Nowatzki. "DAEGEN: A Modular Compiler for Exploring Decoupled Spatial Accelerators". In: *IEEE Computer Architecture Letters* 18.2 (2019), pp. 161–165. DOI: 10.1109/LCA.2019.2955456.

[5] **Jian Weng=**, Sihao Liu=, Vidushi Dadu, Zhengrong Wang, Preyas Shah, and Tony Nowatzki. "DSAGEN: Synthesizing Programmable Spatial Accelerators". In: *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 2020, pp. 268–281. DOI: 10.1109/ISCA45697.2020.00032.

[6] **Jian Weng**, Sihao Liu, Dylan Kupsh, and Tony Nowatzki. "Unifying Spatial Accelerator Compilation With Idiomatic and Modular Transformations". In: *IEEE Micro* 42.5 (2022), pp. 59–69. DOI: 10.1109/MM.2022.3189976.

[7] **Jian Weng**, Sihao Liu, Zhengrong Wang, Vidushi Dadu, and Tony Nowatzki. "A Hybrid Systolic-Dataflow Architecture for Inductive Matrix Algorithms". In: *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2020, pp. 703–716. DOI: 10.1109/HPCA47549.2020.00063.