Research Statement

Tiark Rompf

January 1, 2020

Generative Programming. The main theme of my research explores a radical rethinking of the role of high-level and low-level programming languages under the paradigm of *generative programming*: the central idea is to focus the abstraction power of high-level languages on composing pieces of low-level code, making runtime code generation and domain-specific compiler optimization a fundamental part of the system logic. A key thrust of this work is to build frameworks and tools that make generative programming easy and natural to use, blending in with well-known object-oriented and functional programming paradigms. A cornerstone is the LMS (Lightweight Modular Staging) platform, which provides code generation facilities through pervasive use of operator overloading coupled with sophisticated compiler optimizations. As my work demonstrates, the use of high-level languages and programming patterns in this way vastly reduces the effort required to build performance- and safety-critical systems, and still permits end-to-end bare-metal execution of entire data paths.

Impact. My work has found applications in architecture, databases, machine learning, and security. LMS is directly used by a variety of research groups around the world, and the underlying ideas have influenced numerous other developments. Of particular note is the Delite compiler framework developed with Kunle Olukotun's group at Stanford. OptiML, a domain-specific language based on Delite and LMS featured many of the highlights of today's ML systems like TensorFlow and PyTorch already in 2011, including "in-graph" control flow, operator fusion, transparent distributed multi-GPU execution, etc. This work has further lead to compiler strategies for reconfigurable hardware and the founding of AI chip startup SambaNova Systems for which I serve as a scientific advisor.

Machine Learning and AI. The new AutoGraph front-end in TensorFlow 2.0, the result of a collaboration with Google Brain, is directly based on LMS ideas adapted from Scala to Python, reconciling a convenient imperative programming interface with extraction of computational graphs for optimization and targeting of GPU and TPU accelerators. My group also works on fundamental research with the goal of building deep learning systems that surpass the current generation in expressiveness, based on sound PL foundations: in particular, we have connected backpropagation to continuation-passing, a well-studied program transformation. Lantern, developed by my group, is the first of a new generation of AI systems that can *efficiently* train arbitrary parameterized programs by gradient descent, not just traditional neural networks organized in layers (ICLR'18, NeurIPS'18, ICFP'19).

Data Management Systems. A related focus has been on data management systems, especially query-to-native compilation. We have shown that generative programming makes it possible to compile relevant subsets of SQL to C in just 500 lines of code (ICFP'15), and with only slightly more effort, query compilers can be built for full SQL that rival the best existing engines (SIGMOD'18). Based on these result, my group has built Flare (OSDI'18), a drop-in accelerator for Apache Spark that exhibits 10x-100x speedups on standard benchmarks and enables end-to-end compiled pipelines even across systems, when Spark is used together with TensorFlow. Flare is currently in private beta, attracting significant interest from industry. For this and other contributions, I received the VMware Systems Research Award in 2018, as the third awardee after Tim Kraska (MIT, 2017), and Matei Zaharia (Stanford, 2016).

Security and Verification. In a further line of work, we have demonstrated the utility of generative programming for end-to-end verification: in addition to low-level code, a system can emit contracts and specifications that enable formal verification using standard tools. A recent case study includes a HTTP parser written in 200 lines of Scala, which is as fast as standard web servers such as Apache and Nginx, yet fully verified for memory safety. This result is significant, as all popular web servers suffered from documented security vulnerabilities related to HTTP parsing at various points in the past (POPL'17).

Highest-Performance Computing Kernels. Kernels of mathematical nature such as BLAS or FFT are omnipresent in many domains, and immense tuning effort is necessary to achieve the best performance across platforms. Hence, automatically generating highest-performance implementations from mathematical descriptions is an attractive area for program generation. With Markus Püschel's group at ETH Zürich, I have helped re-implement core parts of the Spiral program generation system, one of the trailblazers in this field. This line of work has lead to new foundational techniques for structuring higly flexible program generators based on parametric types and type classes, an approach summarily described as "generic programming in space and time" (GPCE'17).

Programming Language Foundations. Several lines of my work are focused on core PL foundations and theory. My most significant result in this space is the fully mechanized proof of type soundness for the Dependent Object Types (DOT) calculus, which models the core of Scala's type system (OOPSLA'16, with Nada Amin). Despite significant efforts by several people, this had been an open problem for almost a decade. I am named as a key contributor on the ACM SIGPLAN PL Software Award citation that recognizes the Scala project. Further results include strong normalization for DOT (ECOOP'17), soundness proof techniques based on definitional interpreters (POPL'17), effect systems based on object capabilities (OOPSLA'16), type-directed CPS transformation (ICFP'09), collapsing towers of interpreters (POPL'18), refunctionalization of abstract abstract machines (ICFP'18), and semantic models based on structured time, structured heaps, and collective operations for increased precision in static analysis (OOPSLA'19).

Outlook: Generate All The Things! The switch to flavors of generative programming across systems in industry appears inevitable, and is already under way. A distinction between control and data paths exists in most systems. Leveraging this distinction to add specialized code generation is a natural next step and more and more a necessity due to the changing hardware landscape. All major database vendors are exploring forms of SQL-to-native compilation, programmable data planes are a reality in software-defined networking (SDN), and the eBPF API in the Linux kernel lets user-space programs submit instruction sequences for an in-kernel VM instead of performing series of individual syscalls. All these developments require systems to integrate compiler functionality to stay competitive, which exponentially increases complexity and error surface when implemented using standard low-level techniques. My research provides an avenue to make the development of such embedded compilers an order of magnitude easier, as already demonstrated for SQL databases and other systems.

Systems for AI, AI for Systems. The deep learning and AI space in particular depends on extracting computation graphs from Python code. This is a great opportunity for PL and compiler research to play a leading role in the next wave of co-designed hardware/software systems for AI, ensuring that such systems will be designed based on sound and principled foundations. The need to surpass traditional neural network architectures and to develop systems like Lantern has been prominently proclaimed by Yann LeCun as "Deep Learning est mort. Vive differentiable programming!" In ongoing work, my group is also exploring the reverse direction, using machine learning methods to guide compiler transforms, query optimization, and, in general, symbolic optimization and rewriting problems. We have initial results from a student's research internship at DeepMind, and a recent seminar I taught on Deep Learning for Symbolic Reasoning has lead to several other new and exciting research avenues in this space.

The PurPL Center. On an even broader scale, deep changes are underway in the construction of computer software, with a big shift away from hand-coded rules toward algorithms that are learned from data, sometimes described as Software 2.0. At the same time, software must be built from the start with resilience against adversarial actors in mind, and efficiency considerations demand a shift toward novel hardware architectures, which are hard to program. PL research cannot solve these problems alone, but it will inevitably be a key part of the solution. To address these problems on a broader scale, I have co-founded and am serving as director of a new research center, the Purdue Center for Programming Principles and Software Systems (PurPL). The center is supported by large government grants from various agencies and an industry consortium including Facebook, Microsoft, and SambaNova Systems. Eighteen affiliated faculty cover domains spanning AI, machine learning, security, cryptography, with a proven track record of successful collaboration and tech transfer. Cross-disciplinary projects in which I have leadership roles include secure multi-party computation (IARPA HECTOR) and deep learning for symbolic reasoning (NSF FMitF).