Portal: Portable Accelerated Learning

Our ability to scale computing is facing its biggest challenge in its history. As the performance gains (in terms of energy/op, $/op, and ops/sec) from technology scaling have nearly stopped, people are turning to domain-specific hardware accelerators to create higher performance and more energy-efficient computing systems. This trend is clearly seen in machine learning (ML). Yet it takes a huge investment and several years for domain-specific accelerators to be deployed at scale, because of the enormous engineering effort required to build and verify a robust and performant software system, including an application compiler. Because of the complexity of this task, many accelerator teams produce brittle compilers or only export libraries that exploit the hardware with no automated optimization. Even writing these high-performance libraries is a difficult and time-consuming task.

This center will address the challenge of enabling agile hardware/software development for the large data problems of today and the future. We will demonstrate the capabilities of our approach by creating accelerators and software systems for ML applications.

Four key insights drive our approach. First, all hardware acceleration relies on exploiting parallelism, locality, and specialization to improve application performance. To get sufficient gains to warrant customized hardware, the application must have abundant parallelism: it must be working on problems that deal with large collections of data. Hence, we will tailor our system to creating hardware/software to accelerate applications that work on large data collections.

Second, creating performant and energy-efficient applications always requires tuning of the algorithms, software and hardware. So, our approach must provide ways of measuring the performance and energy of the hardware on an application and provide these measurements in a manner that a programmer would understand and can use to perform tuning.

Third, it is hard and expensive to build complex systems. We must use agile methods to address this issue that largely reuse and incrementally modify existing hardware and software systems, rather than create a new system from scratch. So future hardware systems must integrate accelerators on existing base systems. Yet this doesn't resolve the software issue: creating a custom compiler for each application-domain/accelerator pair requires too much effort and prevents interoperability. It is often essential for the base system to be able to connect different types of computational flows together and to different accelerators. This task is usually accomplished by a common IR used to represent all the different flows.

Finally, the LLVM IR, which is used to compile most code today, is excellent at expressing and optimizing the execution of primitive operations, but is much too low-level to interface to accelerators that perform complex operations on data collections. Its primitives need to be lifted into operations on tiles and other small collections, which is a difficult task.

Approach

We believe four abstract data models (abstract collections), and associated operations, cover most things we are currently computing on at scale: (1) tensors, (2) graphs, (3) relations, and (4) space. While similar, these data models are useful for expressing different things at the application level. For example, a simulation may store a mesh in a graph, assemble global stiffness matrices from the graph, and assemble collision resolution matrices from a collision-detection query over space. And a recommender model may perform a k-nearest neighbor query over a high-dimensional space followed by tensor operations over

embeddings. Tensors are also the fundamental building blocks of many deep learning libraries, which have to navigate the tension between flexibly supporting myriad network architectures and achieving high hardware utilization for each architecture.

We plan to build programming models around these abstractions, with language constructs to transition between different types of collections (e.g., from a graph to a matrix). With appropriate operations (e.g., tensor algebra and set queries), we can make sure the applications are detached from the order of operations and where data is stored, making it possible to bind these decisions in the compiler and hence get portability. Our prior work in creating TACO, a compiler for portable dense and sparse tensor algebra will guide this work. There are many popular languages that already operate at the data collection level, such as pandas and PyTorch. As a starting point, we will use PyTorch with sparse extensions (so it can naturally support models like graph neural networks and mixture of experts) as the front-end language for our ML compiler.

We can reduce the operations on the four collections to one unified IR that is capable of expressing how we traverse and co-traverse data structures, enabling one infrastructure that optimizes and merges code across the different collections (e.g., fuse tensor algebra onto dataframe data structures). We will use this single IR framework to provide portability across heterogeneous accelerators by requiring the accelerators to register their “capabilities” with this compiler. These capabilities would be expressed in terms of a capability language that enumerates what operations in the application languages (or the IR) they support, along with the cost (time, energy) of the operations, enabling the compiler to optimize the performance and/or energy of the application.

To provide concrete test cases for our compiler, we will explore different types of hardware accelerators. Many of these will come from an ML accelerator generator that we will create. This tool will explore efficient methods for hardware design such as high-level synthesis, as well as approaches for making accelerators more energy-efficient. It will also serve as a testbed for experimenting with the compiler and evaluating the completeness of the capability language. Ultimately, we will extend this accelerator generator to create the backend compiler needed to map the high-level collection operations to the accelerator’s native ISAs or configuration. In addition to the work on this generator, we plan to explore some non-traditional computing approaches to these problems to both cast a wider net and to stress test our software system. We plan to explore possible acceleration options for a range of compute models, including hyperdimensional computing, p-computing, and digital-amenable analog computing models, such as oscillator-based computing. We also will explore inverse design, data-driven model augmentation, and analysis of materials and circuits for analog (and quantum) computation.

Our prior work has also shown the advantage of creating generators which can automatically create formal models of the generated hardware. We will use this approach to help us automate design tasks and validate the hardware. We will also leverage formal techniques for non-functional validation and for ensuring the correctness of transformations performed by the compiler. We will explore strategies for improving and configuring our tools, including our hardware generator and our compiler, in order to improve our ability to leverage formal analysis.

Finally, creating energy-efficient and performant systems always requires application/hardware co-design. For example in ML, while high-level libraries such as Jax or PyTorch make it possible to express new model architectures, the resulting code often does not achieve high hardware utilization without careful optimization by hand. This optimization is a time-intensive process that requires specialized knowledge about the underlying accelerators, and slows down model innovation. An essential part of our research is to address this issue which includes creating a compiler/simulator which is able to generate and track the right performance/energy/area/accuracy metrics critical for application optimization. Next, we must build

our IR and compiler system with the right abstractions to allow these metrics to be presented to the application programmer in a manner which they can understand. Finally, we will create optimization techniques for recurring problems to further automate this task. For example, quantization to low precision numerics is important for achieving higher op/W. However, their selection and deployment involves making careful accuracy-efficiency tradeoffs by exploring a large space of design decisions. Our goal for a common issue like this is to not only provide appropriate feedback, but to create systems which automate the application-hardware/software design optimization, making deployment of ML applications using fine-grained low/mixed-precision formats easier.

Faculty

Portal brings together eight faculty from Stanford University—Mark Horowitz, Thierry Tambe, Priyanka Raina, Fred Kjolstad, Sara Achour, Clark Barrett, Mert Pilanci, and Ludwig Schmidt—with expertise spanning hardware accelerators, compilers, programming languages, formal methods and machine learning, including both ML algorithms, and model compression and quantization techniques. The faculty have extensive experience in building real research prototypes, several of which have turned into successful commercial technologies.

Mark Horowitz is the Fortinet Founders Chair of Electrical Engineering and the Yahoo! Founders Professor in the School of Engineering. He co-founded Rambus, Inc. in 1990 and is a fellow of the IEEE and the ACM and a member of the National Academy of Engineering and the American Academy of Arts and Science. His research interests are quite broad and span from applying EE and CS analysis methods to problems in molecular biology, to creating new design methodologies for analog and digital VLSI circuits.

Thierry Tambe is an Assistant Professor of Electrical Engineering at Stanford University. His research interests include hardware and software co-design techniques for domain-specific silicon systems for emerging AI and compute/memory-intensive applications. Previously, Thierry was an engineer at Intel where he worked on mixed-signal architectures for high-bandwidth memory and peripheral interfaces on Xeon HPC SoCs. He received a B.S., and M.Eng. from Texas A&M University, and a PhD from Harvard University. He is a recipient of a NVIDIA Graduate PhD Fellowship, an IEEE SSCS Predoctoral Achievement Award, and distinguished paper awards at DAC and MICRO.

Priyanka Raina is an Assistant Professor of Electrical Engineering at Stanford University. She works on domain-specific hardware accelerators and agile hardware–software codesign methodologies, particularly for machine learning. She received her B.Tech. degree from IIT Delhi, and M.S. and PhD degrees from MIT. Her research has won best paper awards at VLSI, MICRO, ESSCIRC and in JSSC. She has won the Sloan Research Fellowship, the NSF CAREER Award, the Intel Rising Star Faculty Award, the Hellman Faculty Scholar Award and is a Terman Faculty Fellow.

Fredrik Kjolstad is an Assistant Professor of Computer Science at Stanford University. He works on compilers, programming models, and systems, with an emphasis on portability. His research includes the TACO Sparse Tensor Algebra Compiler and the Simit language for computing on sparse systems. He has received the NSF CAREER Award, the MIT Sprowls PhD Thesis Award in Computer Science, the Rosing Award, an Adobe Fellowship, a Google Research Scholarship, and several best/distinguished paper awards.

Sara Achour is an Assistant Professor jointly appointed to both the Computer Science and the Electrical Engineering departments at Stanford University. Her research focuses on new techniques and tools, specifically new programming languages, compilers, and runtime systems, that enable end-users to more easily develop computations that exploit the potential of emerging computing platforms that exhibit analog behaviors.

Clark Barrett is a Professor (Research) of Computer Science at Stanford University. His expertise is in automated reasoning and its applications. He was an early pioneer in formal hardware verification as part of 0-in Design Automation (now a part of Siemens), where he helped build one of the first successful formal verification toolsets for hardware. His current work focuses on the development and application of automated reasoning techniques to improve reliability and security of software, hardware, and ML systems. He is an ACM Distinguished Scientist and a winner of the 2021 Computer Aided Verification award.

Mert Pilanci is an Assistant Professor of Electrical Engineering at Stanford University. He received his Ph.D. in Electrical Engineering and Computer Science from UC Berkeley in 2016. Prior to joining Stanford, he was an Assistant Professor of Electrical Engineering and Computer Science at the University of Michigan. In 2017, he was a Math+X postdoctoral fellow working with Emmanuel Candès at Stanford University. His research interests are in large scale machine learning, neural networks, optimization, and information theory.

Ludwig Schmidt is an incoming Assistant Professor of Computer Science at Stanford University. Ludwig's research interests revolve around the empirical foundations of machine learning, often with a focus on datasets, reliable generalization, and large models. Recently, Ludwig's research group contributed to open source machine learning by creating OpenCLIP, DataComp, and the LAION-5B dataset. Ludwig completed his PhD at MIT and was a postdoc at UC Berkeley. Ludwig's research received best paper awards at ICML and NeurIPS, a best paper finalist at CVPR, and the Sprowls dissertation award from MIT.