# Research Statement

Shoaib Kamil
skamil@mit.edu

As clock speed scaling has slowed down, processor manufacturers have shifted to increasing hardware parallelism to maintain Moore's Law, and we can no longer rely on yearly processor speed improvements to increase performance. Nor can we rely purely on improvements in compilers. Compiler optimizations on their own have a poor track record: as Proebsting [TAB14] observes, compilers for general purpose languages have only increased performance by 4% per year. At the same time, we know highly-skilled programmers have been able to increase performance by carefully writing hand-optimized low-level parallel code, taking advantage of optimizations that do not apply to all programs. To continue increasing performance, the path forward is to encode this knowledge in automatic domain-specific systems that parallelize and optimize programs.

My research focuses on optimization techniques and programming systems (programming languages, compilers, and runtimes) that take advantage of domain-specific high level information to deliver improved performance and programmability. The goal is to improve the building blocks programmers use, in order to make programming more efficient, both in terms of time writing code and in terms of delivered performance. Trends in bleeding-edge platforms, including the proliferation of accelerators and new programming models such as probabalistic and unreliable computing, make this goal even more urgent.

My approach is to first look at how performance experts optimize code for a particular kind of computation; generalize, codify, and understand the impact on performance; and then develop tools that automate these optimizations for the domain where they apply. These tools are most often domain-specific languages (DSLs) and compilers. The aim is to sidestep Proebsting's Law by foregoing general purpose optimizations and instead concentrating on specific kinds of computation. To ensure code written in general purpose languages can benefit from domain-specific optimizations, my newest area of research is sound automated methods to transform from general purpose languages to DSLs. This transformed code can take advantage of the power of DSL compilers to obtain performance.

In the course of applying this approach, I have developed three core technologies that make this research approach possible.

## Core Technology

**SEJITS: High Performance DSLs for Dynamic Languages** Many programmers prefer to work in high-level dynamically-typed languages such as Python and Ruby, due to their high expressiveness and the availability of powerful libraries. However, their performance can be far worse than low-level languages like C++ or Fortran. I helped create Selective Embedded Just-In-Time Specialization (SEJITS), a methodology for constructing high performance DSLs in high-level interpreted languages [PMEA09], and Asp, a library for building such DSLs in Python [UCB13, PPOPP12, SCIPY11]. DSLs using this approach internally transform code into fast, parallel implementations that outperform Python by over $1000\times$. The key idea is to sidestep the interpreter, and instead use the computation defined by the programmer as a declarative specification for a program that performs the same computation using low-level parallel constructs in C++, Scala, or CUDA. Asp also generates code that passes needed data from Python to the parallel implementation and back. This program and glue code is automatically compiled & run behind the scenes by Asp in place of the original Python code. As a result, when a programmer runs their program, it appears to simply run much faster than it would normally; the entire process is invisible to users. Asp makes it much easier to build embedded DSLs using the SEJITS approach, which continues to be a major thrust of ongoing research at UC Berkeley.

**OpenTuner** A key enabling technology for much of my research is auto-tuning, which uses empirical search over a space of correct implementations of a program to find the best-performing version. I conceived and helped supervise OpenTuner, a framework for building domain-specific multi-objective auto-tuners [PACT14]. Prior to OpenTuner, most users of auto-tuning were applying either exhaustive search or simplistic algorithms such as hill climbing. This made auto-tuning over large spaces intractable. In contrast, OpenTuner packages sophisticated machine learning techniques along with a customizable representation of the search space, making it much easier to build powerful auto-tuners. A key idea in OpenTuner is to combine multiple search techniques using a multi-armed bandit which asssigns more resources to those techniques that find better solutions; this enables combining the best of multiple general and domain-specific search techniques. OpenTuner is being used in a number of projects, including domain-specific compilers and languages for stream programming.

**Raising the Level of Abstraction With Synthesis** My most recent research thrust aims to bridge the gap between general purpose and domain-specific languages by building systems that automatically infer high-level programmer intent from low-level code. The key enabling technology for this is program synthesis [SKETCH], which efficiently searches a space of candidate implementations to find one that satisfies a given specification. To automatically transform snippets of domain-specific code from general languages, we construct verification conditions for the snippet, leaving the postcondition as an unknown function. We then use synthesis to search over a space of candidate postconditions, each of which fits into the domain in question, to find one that makes verification succeed— this allows us to infer that the programmer's semantic intent. We can then replace the snippet of code with any other code that results in the same postcondition, including code in domain specific languages.

# Selected Current Research

These core technologies have enabled me to pursue a set of research projects that span all aspects of domain-specific program optimization. Here, I highlight some of my current research projects.

**Optimizations & Domain-Specific Compilers for Stencils** Stencil computations constitute an important computational pattern that is used in scientific simulations, linear solvers, image processing algorithms, and machine learning. In such computations, each point in a multidimensional space is updated with a function of its neighbors.

We explored existing and new optimizations for stencils on modern architectures [MSP05, MSPC06, CF06, SIREV09] and found that many traditional optimizations were ineffective due to more complicated memory hierarchies in modern computers. Based on the optimizations we discovered, we built the first modern parallelizing stencil compiler [CUG09, IPDPS10] that targeted multicore CPUs and GPUs. With this compiler, programmers could express their computation in a simple, straightforward language, and obtain over 90% of peak performance across architectures. The compiler utilized domain-specific transformations as well as empirical auto-tuning— it generated many correct variants of the code and ran them to find the most optimal. This approach is the basis for modern high performance stencil DSLs, including PATUS [PATUS], which is the successor to our original work, and Halide [HALIDE], which is used in production at Google and other corporations.

**Embedded Domain-Specific Languages in Python** Using Asp and the SEJITS methodology, I have helped build domain-specific compilers for machine learning [HOTPAR11, SCIPY12], stencil computations [UCB13], communication-avoiding matrix algorithms [IPDPS13A, IWAPT12], and queries over extremely large-scale graphs [IPSPDS13B, JPDC14]. These compilers target GPUs, multicore CPUs, and clusters, and obtain much higher performance than is possible with pure Python. In a number of cases, the performance is equivalent to or better than programs hand-written by performance experts in low-level languages, due to the use of auto-tuning. Additional SEJITS DSLs are in active development at UC Berkeley.

**Automatically Transforming Stencils in Fortran to DSLs** Using our new synthesis-based technology for bridging between general languages and domain-specific languages, we are building STNG, a system [SUB1] that transforms code automatically from Fortran to Halide. Preliminary results show that this approach results in speedups of over $10\times$, outperforming existing fully-automatic approaches such as polyhedral analysis. In order to build STNG, we overcame a number of obstacles, including transforming a large triply-quantified boolean formula into one that could be solved efficiently using program synthesis. The system is able to transform stencil computations from real applications used in scientific computing and image processing.

Our technique has also been applied to transform database queries in Java, with similarly excellent results [QBS]. In both cases, the programmer does not need to write or change any of their code by hand; the system automatically and soundly replaces the code written in a general language with new code in a DSL that does the same computation.

**Creating Distributed Implementations from Serial Specifications** When data is partitioned across several machines, it becomes much more complicated to reason about computations and data transformations, due to the need to communicate and synchronize data. I have been involved in building MSL [SC14], a new language for synthesizing parallel implementations from serial specifications. In MSL, users provide a serial implementation plus two things: a description of how the data is distributed, and a space of possible parallel implementations. The synthesizer in MSL finds a parallel implementation that performs the same computation as the serial code and outputs C++ code using MPI for communication. Kernels built using MSL perform as well as those written by hand in Fortran. This ongoing project has the potential to make it much easier to write complicated distributed computations.

# Future Directions

For future explorations, I plan to continue building new domain-specific compilers and languages, as well as new enabling infrastructure. In addition, I will explore new problems arising as parallelism and the need for optimization become more ubiquitous.

**Bridging General Purpose and Domain-Specific Compilers** I believe there is huge potential in using program synthesis to bridge between domain-specific and general purpose languages. Though this has been demonstrated for two kinds of computations (stencils and database queries), many other computations can potentially benefit from this approach. For stencils, much of the work was making it tractable to solve a complex synthesis problem with additional quantifiers; it is likely new difficulties will arise as we build this technology for other domains. Given the potential payoff (orders of magnitude performance improvements, automatically), this is an exciting area of research. The first additional area I will look into is computations on irregular grids, which are difficult for polyhedral tools to optimize. Based on my experience with STNG, I believe these computations will be possible to transform using this approach.

**Supporting Parallelism in Compilation** Current low-level intermediate representations such as LLVM do not directly represent parallelism. Instead, the current approach is that parallelism is handled by front-end compilers; by the time the code is transformed into the low-level representation, all decisions about parallelism have already been made. This can result in sub-optimal performance when composing parallelism, and makes it difficult to generate multiple parallelism strategies which could be auto-tuned.

I want to explore an alternative approach, where parallelism *decisions* are left to the low-level representation. To be clear, this is not automatic parallelization; at this level, the code includes all available parallelism up to the finest granularity, while decisions about how to exploit parallelism occur in the backend. This would enable the generation of multiple variants at the backend level, and could, if combined with parallel optimizations at this level, make it easier to generate optimized parallel code. With the increasing importance of parallel execution, pushing parallel decisions into the low-level could make it simpler and easier to obtain high performance. Early exploratory results show that this approach works better than the current approach for Cilk-style fork-join parallelism. The next step is to see how other shared-memory models can be optimized in this approach, and then to extend it to distributed computations. The goal is to support many different parallel and distributed programming models with a unified low-level representation.

**New Domain Specific Optimizers** Large-scale datacenters and mobile devices are two of the driving forces for new areas of computing. I am exploring web applications as one of the areas where the domain-specific approach optimization can be effective. Many web applications are written in high-level interpreted languages such as Ruby, Python, and JavaScript; making such applications more efficient could greatly impact current and future computing. I have already begun exploring some optimization techniques that seem promising for these important applications. If these optimizations can be generalized, then we can begin exploring how to either leverage existing frameworks (such as Rails) or to build new ones that enable optimization.

In addition, I'm looking forward to collaborating with new colleagues to find what they think are the most important kinds of computations in their applications. Across many subjects, computation is becoming a central piece of research, and for many of these areas, we do not yet understand the most important bottlenecks that can be eased through the use of domain-specific optimization techniques.

**Mining Software for Parallelism & Synthesis** With the proliferation of open source repositories on sites like Github and Bitbucket, we now have a large corpus of existing programs that could be analyzed. In addition, programmer-oriented sites like StackOverflow help explain difficult programming questions in a combination of prose and program snippets. I am interested in leveraging the knowledge contained in these repositories using new machine learning and program analysis techniques. Potentially, we can use these repositories as a basis for learning new automated or semi-automated program transformation techniques. For example, some early work I've supervised has shown it is possible use ML techniques to learn how to transform code to use new versions of a library.

Using historical information from source control check-ins, I plan on beginning by seeing if transformations from serial to parallel code can be discovered, and perhaps generalized. Other kinds of transformations may be useful as high-level templates, using program synthesis to ensure correctness. If these new sources of code and programmer knowledge can be leveraged to improve programming tools, the impact will be huge.

## Summary

In a decade, the landscape of compilers and languages will have fundamentally changed in response to the need for parallelism to obtain performance and the proliferation of new execution models and architectures. Programmers will write code in combinations of domain-specific languages and general purpose languages. These programs will express high-level code without low-level optimizations; domain-specific compilers and general purpose languages with domain-specific optimizers will be responsible for turning this code into high performance, energy-efficient executables that will run on a wider variety of architectures, taking advantage of new programming models such as approximate and probabilistic execution.

My research brings us closer to this future, by building programming systems that improve both productivity and performance. I believe the path forward is to exploit domain-specific information to guide optimization strategies, and to build programming systems that allow programmers to express their programs without conflating the desired computation with low-level optimizations. As parallelism becomes even more ubiquitous, my research will continue to enable programmers to program with higher productivity and higher performance.

## Works Cited

[SUB1] *Automated Synthesis-Driven Optimization of Stencil Codes*. Shoaib Kamil, Alvin Cheung, Armando Solar-Lezama. Submitted.

[SC14] *MSL: A Synthesis-Enabled Language for Distributed Implementations*. Zhilei Xu, Shoaib Kamil, Armando Solar-Lezama. Supercomputing 2014

[PACT14] *OpenTuner: An Extensible Framework for Program Autotuning*. Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Una-May O'Reilly, Saman Amarasinghe. Parallel Architectures and Compilation Techniques (PACT), 2014

[JPDC14] *Parallel Processing of Filtered Queries in Attributed Semantic Graphs*. Adam Lugowski, Shoaib Kamil, Aydin Buluc, Samuel Williams, Erika Duriakova, Leonid Oliker, Armando Fox, John Gilbert. Journal of Parallel and Distributed Computing (JPDC), 2014

[UCB13] *Productive High Performance Parallel Programming with Auto-tuned Domain-Specific Embedded Languages*. Shoaib Kamil. PhD Thesis, University of California, Berkeley

[IPDPS13A] *Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication*. James Demmel, David Eliahu, Armando Fox, Shoaib Kamil, Benjamin Lipshitz, Oded Schwartz, Omer Spillinger. International Parallel and Distributed Processing Symposium (IPDPS), 2013

[IPDSP13B] *High-Productivity and High-Performance Analysis of Filtered Semantic Graphs*. Aydin Buluc, Erika Duriakova, Armando Fox, John Gilbert, Shoaib Kamil, Adam Lugowski, Leonid Oliker, Samuel Williams. International Parallel and Distributed Processing Symposium (IPDPS), 2013

[IWAPT12] *Auto-tuning the Matrix Powers Kernel with SEJITS*. Jeffrey Morlan, Shoaib Kamil, Armando Fox. Seventh International Workshop on Automatic Performance Tuning (iWAPT), 2012

[SCIPY12] *Parallel High Performance Statistical Bootstrapping in Python*. Aakash Prasad, David Howard, Shoaib Kamil, Armando Fox. Scientific Computing with Python Conference, 2012

[PPOPP12] *Portable Parallel Performance from Sequential, Productive, Embedded Domain Specific Languages*. S. Kamil, D. Coetzee, S. Beamer, H. Cook, E. Gonina, J. Harper, J. Morlan, A. Fox. ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Extended Abstract, 2012

[SCIPY11] *Bringing Parallel Performance to Python with Domain-Specific Selective Embedded Just-in-Time Specialization*. Shoaib Kamil, Derrick Coetzee, Armando Fox. 10th Python for Scientific Computing Conference, 2011

[HOTPAR11] *CUDA-level Performance with Python-level Productivity for Gaussian Mixture Model Applications*. H. Cook, E. Gonina, S. Kamil, G. Friedland, D. Patterson, A. Fox. USENIX Workshop on Hot Topics in Parallelism (HotPar), 2011

[IPDPS10] *An Auto-tuning Framework for Parallel Multicore Stencil Computations*. Shoaib Kamil, Cy Chan, Leonid Oliker, John Shalf, Samuel Williams. IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2010

[PMEA09] *SEJITS: Getting Productivity and Performance with Selective Embedded JIT Specialization*. Bryan Catanzaro, Shoaib Kamil, Yunsup Lee, Krste Asanovic, James Demmel, Kurt Keutzer, John Shalf, Kathy Yelick, Armando Fox. Workshop on Programming Models for Emerging Architectures (PMEA), 2009

[CUG09] *A Generalized Framework for Auto-tuning Stencil Computations*. Shoaib Kamil, Cy Chan, Sam Williams, Leonid Oliker, John Shalf, Mark Howison, E. Wes Bethel, Prabhat. Cray User Group Conference, 2009 Best Paper Award

[SIREV09] *Optimization and Performance Modeling of Stencil Computations on Modern Microprocessors*. Kaushik Datta, Shoaib Kamil, Sam Williams, Leonid Oliker, John Shalf, Katherine Yelick. SIAM Review, 2009

[MSPC06] *Implicit and Explicit Optimizations for Stencil Computations*. Shoaib Kamil, Kaushik Datta, Samuel Williams, Leonid Oliker, John Shalf, Katherine Yelick. Memory Systems Performance and Correctness (MSPC), 2006

[CF06] *The Potential of the Cell Processor for Scientific Computing*. Sam Williams, John Shalf, Parry Husbands, Shoaib Kamil, Leonid Oliker, Katherine Yelick. ACM International Conference on Computing Frontiers, 2006

[MSP05] *Impact of Modern Memory Subsystems on Cache Optimizations for Stencil Computations*. Shoaib Kamil, Parry Husbands, Leonid Oliker, John Shalf, Katherine Yelick. ACM SIGPLAN Workshop on Memory Systems Performance (MSP), 2005

## Works By Others Cited

[HALIDE] *Halide: A Language and Compiler for Optimizing Parallelism, Locality, and Recomputation in Image Processing Pipelines*. Jonathan Ragan-Kelley, Connelly Barnes, Andrew Adams, Sylvain Paris, Frdo Durand, Saman Amarasinghe. Programming Language Design and Implementation (PLDI), 2013

[PATUS] *PATUS: A Code Generation and Autotuning Framework for Parallel Iterative Stencil Computations on Modern Microarchitectures*. M. Christen, O. Schenk, H. Burkhart. International Parallel and Distributed Processing Symposium (IPDPS), 2011

[SKETCH] *Combinatorial Sketching for Finite Programs*. Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Vijay Saraswat, Sanjit A. Seshia. Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2006

[TAB14] Todd A Proebsting's Page. *http://proebsting.cs.arizona.edu*.