Finding a Research Topic & Interdisciplinary Research

Armando Solar-Lezama, CSAIL, MIT
Gonzalo Ramos, Microsoft Research

based on Melanie E. Moses' slides

2019 CRA URMD Grad Cohort Workshop

CRA
Computing Research Association

Armando Solar-Lezama

- Moved to the US with my family from Mexico when I was 15
- BS from Texas A&M University, PhD from UC Berkeley
- Faculty at MIT since 2008
- Work on Programming Systems + X

CRA
Computing Research Association

Gonzalo Ramos

- From Argentina
- MS, PhD in CS, University of Toronto, Canada
- Scientist, Microsoft Live Labs; OSD (2007-2013)
- Sr Design Technologist - Amazon Concept Lab (2013-2015)
- Sr UX Scientist - Amazon Grand Challenges (2015-2016)
- Sr Researcher - Microsoft Research AI

CRA
Computing Research Association

The Finding Path

Finding & Changing...

"Find a place you trust, and then try trusting it for awhile."

John Cage and Sister Corita Kent

- Identifying a research topic is equal parts asking questions about the world, and yourself.
- It is often does not resemble a straight line.
- Do investments, Also be ready to change.
- It is ultimately personal, not two are the same

CRA
Computing Research Association

Research Journeys...

- The uncompromising vision.

  “My goal is (grand and) clear. My steps unwavering.”

- The opportunistic explorer.

  “I see an array of possibilities. I choose and change when appropriate.”

- The problem solver.

  “I see something I know how to fix. I learn by doing.”

- Yours!

  “your story goes here...”

CRA
Computing Research Association

Strategies and Learning to Ask Questions

Don't try to create and analyze at the same time. They're different processes.

John Cage and Sister Corita Kent

Finder's Strategies

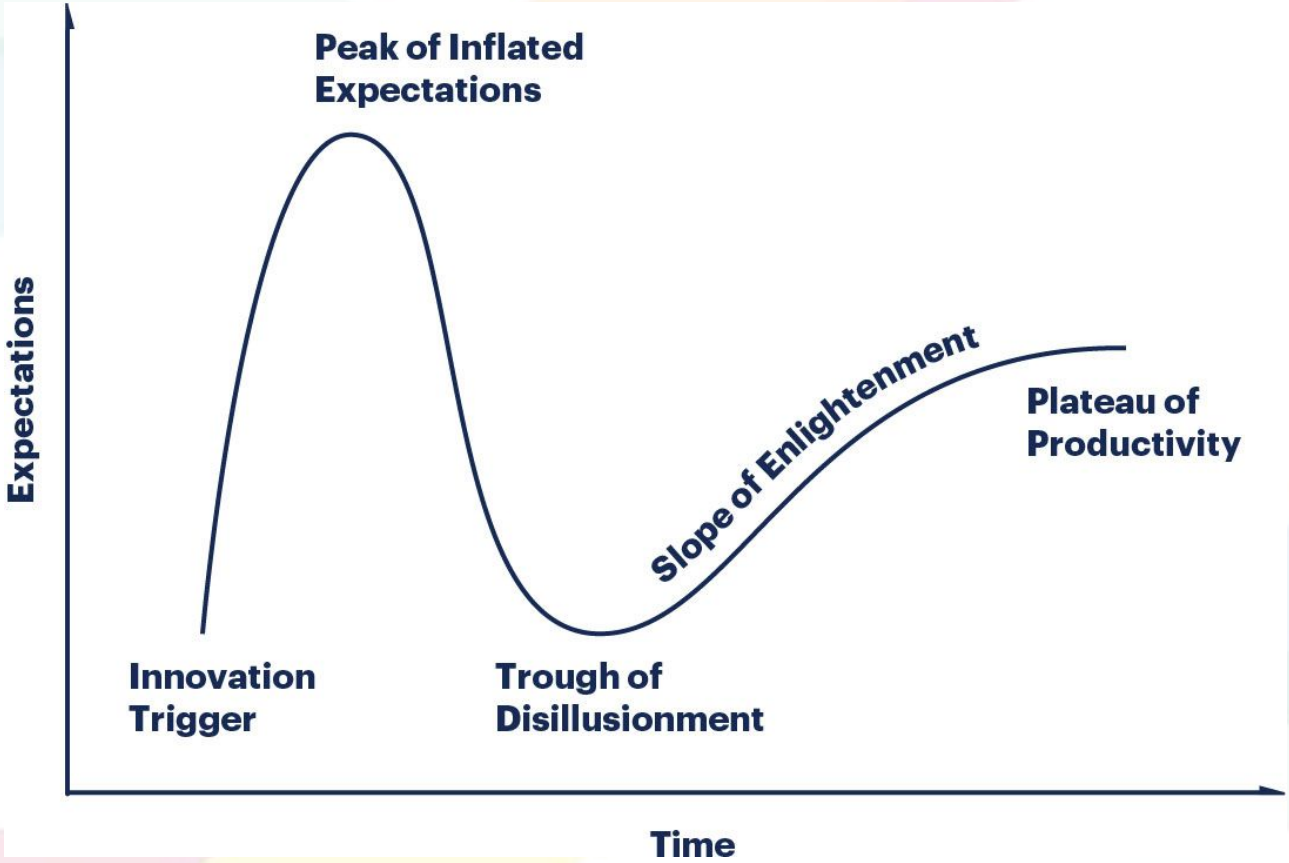
"Always be around. Come or go to everything. Always go to classes. Read anything you can get your hands on. Look at movies carefully, often. Save everything."

John Cage and Sister Corita Kent

- What are you passionate about?
- What are your values?
- How do these guide you & fit?

CRA
Computing Research Association

Beware of the Hype...



https://www.gartner.com/en/research/methodologies/gartner-hype-cycle

CRA
Computing Research Association

Tips Along the Way

- The journey of a student is individual.
- The journey of a researcher is collective.
- Envision success, and work backwards from it into metrics, and milestones.
  - Try the press-release approach.
  - What do you bring to the table?
- Asses resources and knowledge/skills gap

CRA
Computing Research Association

Interdisciplinary Research

Finding Interdisciplinary Topics

- In my field (HCI), almost all work is interdisciplinary.
- Go out and learn about...
  - people and their problems.
  - a new technology that has not been applied before.
  - what you can give.
- Take advantage of the challenge of being not exactly where you planned.
- Try explaining what you do to others in different fields, learn their perspective.

CRA
Computing Research Association

Advising Interdisciplinary Topics

- One, two or more advisors and mentors?
- Do they appropriately balance breadth vs depth of research?
- Do they have a core identity that supports or overlaps with yours?
- Are they open-minded and enthusiastic about learning from other fields?
- Can they provide financial support for interdisciplinary research?
- Will you find a community of researchers that support your work?

CRA
Computing Research Association

Case Studies

Towards Optimization-Safe Systems: Analyzing the Impact of Undefined Behavior

Xi Wang, Nickolai Zeldovich, M. Frans Kaashoek, and Armando Solar-Lezama
MIT CSAIL

Abstract

This paper studies an emerging class of software bugs called *optimization-unstable code*: code that is unexpectedly discarded by compiler optimizations due to undefined behavior in the program. Unstable code is present in many systems, including the Linux kernel and the Postgres database. The consequences of unstable code range from incorrect functionality to missing security checks.

To reason about unstable code, this paper proposes a novel model, which views unstable code in terms of optimizations that leverage undefined behavior. Using this model, we introduce a new static checker called STACK that precisely identifies unstable code. Applying STACK to widely used systems has uncovered 160 new bugs that have been confirmed and fixed by developers.

```
char *buf = ...;
char *buf_end = ...;
unsigned int len = ...;
if (buf + len >= buf_end)
    return;   /* len too large */
if (buf + len < buf)
    return;   /* overflow, buf+len wrapped around */
/* write to buf[0..len-1] */
```

Figure 1: A pointer overflow check found in several code bases. The code becomes vulnerable as gcc optimizes away the second `if` statement [13].

unstable code happens to be used for security checks, the optimized system will become vulnerable to attacks.

This paper presents the first systematic approach for reasoning about and detecting unstable code. We implement this approach in a static checker called STACK, and use it to show that unstable code is present in a wide

Managing Performance vs. Accuracy Trade-offs With Loop Perforation

Stelios Sidiroglou Sasa Misailovic Henry Hoffmann
Martin Rinard
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
{stelios,misailo,hank,rinard}@csail.mit.edu

ABSTRACT

Many modern computations (such as video and audio encoders, Monte Carlo simulations, and machine learning algorithms) are designed to trade off accuracy in return for increased performance. To date, such computations typically use ad-hoc, domain-specific techniques developed specifically for the computation at hand.

Loop perforation provides a general technique to trade accuracy for performance by transforming loops to execute a subset of their iterations. A criticality testing phase filters out *critical loops* (whose perforation produces unacceptable behavior) to identify *tunable loops* (whose perforation produces more efficient and still acceptably accurate computations). A perforation space exploration algorithm perforates combinations of tunable loops to find Pareto-optimal perforation policies. Our results indicate that, for a range of applications, this approach typically delivers performance increases of over a factor of two (and up to a factor of seven) while changing the result that the application produces by less than 10%.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Reliability, availability, and serviceability

This paper presents and evaluates a technique, *loop perforation*, for generating new variants of computations that produce approximate results. These variants occupy different points in the underlying performance vs. accuracy tradeoff space. Our results show that perforating appropriately selected loops can produce significant performance gains (up to a factor of seven reduction in overall execution time) in return for small (less than 10%) accuracy losses. Our results also show that the generated variants occupy a broad region of points within the tradeoff space, giving users and systems significant flexibility in choosing a variant that best satisfies their needs in the current usage context.

1.1 Loop Perforation

Loop perforation transforms loops to execute a subset of their iterations. The goal is to reduce the amount of computational work (and therefore the amount of time and/or other resources such as power) that the computation requires to produce its result. Of course, perforation may (and in our experience, almost always does) cause the computation to produce a different result. But approximate computations can often tolerate such changes as long as they do not unacceptably reduce the accuracy. Our implemented system uses the following techniques to find effective perforations:

A new approach to achieve something that has never been achieved before.

Programming by Sketching for Bit-Streaming Programs

Armando Solar-Lezama[1] Rodric Rabbah[2] Rastislav Bodík[1] Kemal Ebcioğlu[3]

[1]Computer Science Division, University of California, Berkeley
[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology
[3]T.J. Watson Research Center, IBM Corporation
{asolar, bodik}@cs.berkeley.edu, rabbah@mit.edu, kemal@us.ibm.com

Abstract

This paper introduces the concept of *programming with sketches*, an approach for the rapid development of high-performance applications. This approach allows a programmer to write clean and portable reference code, and then obtain a high-quality implementation by simply *sketching* the outlines of the desired implementation. Subsequently, a compiler automatically fills in the missing details while also ensuring that a completed sketch is faithful to the input reference code. In this paper, we develop StreamBit as a sketching methodology for the important class of bit-streaming programs (e.g., coding and cryptography).

A sketch is a *partial* specification of the implementation, and as such, it affords several benefits to programmer in terms of productivity and code robustness. First, a sketch is easier to write compared to a complete implementation. Second, sketching allows the programmer to focus on exploiting algorithmic properties rather than on orchestrating low-level details. Third, a sketch-aware compiler rejects "buggy" sketches, thus improving reliability while allowing the programmer to quickly evaluate sophisticated implementation ideas.

1. Introduction

Applications in domains like cryptography and coding often have the need to manipulate streams of data at the bit level. Such manipulations have several properties that make them a particularly challenging domain from a developer's point of view. For example, while bit-level specifications are typically simple and concise, their word-level implementations are often daunting. Word-level implementations are essential because they can deliver an order of magnitude speedup, which is important for servers where security-related processing can consume up to 95% of processing capacity [20]. Converting bit-level implementations to word level implementations is akin to vectorization, but the characteristics of bit-streaming codes render vectorizing compilers largely ineffective. In fact, widely used cipher implementations often achieve performance thanks to algorithm-specific algebraic insights that are not available to a compiler.

Additionally, correctness in this domain is very important because a buggy cipher may become a major security hole. In 1996, a release of the BlowFish cipher contained a buggy cast (from unsigned to signed characters) which threw away two thirds of the

Good problem selection will only take you so far.

Programming by Sketching for Bit-Streaming Programs

Armando Solar-Lezama[1] Rodric Rabbah[2] Rastislav Bodík[1] Kemal Ebcioğlu[3]

[1]Computer Science Division, University of California, Berkeley
[2]Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology
[3]T.J. Watson Research Center, IBM Corporation
{asolar, bodik}@cs.berkeley.edu, rabbah@mit.edu, kemal@us.ibm.com

Abstract

This paper introduces the concept of *programming with sketches*, an approach for the rapid development of high-performance applications. This approach allows a programmer to write clean and portable reference code, and then obtain a high-quality implementation by simply *sketching* the outlines of the desired implementation. Subsequently, a compiler automatically fills in the missing details while also ensuring that a completed sketch is faithful to the input reference code. In this paper, we develop StreamBit as a sketching methodology for the important class of bit-streaming programs (e.g., coding and cryptography).

A sketch is a *partial* specification of the implementation, and as such, it affords several benefits to programmer in terms of productivity and code robustness. First, a sketch is easier to write compared to a complete implementation. Second, sketching allows the programmer to focus on exploiting algorithmic properties rather than on orchestrating low-level details. Third, a sketch-aware compiler rejects "buggy" sketches, thus improving reliability while allowing the programmer to quickly evaluate sophisticated implementation ideas.

1. Introduction

Applications in domains like cryptography and coding often have the need to manipulate streams of data at the bit level. Such manipulations have several properties that make them a particularly challenging domain from a developer's point of view. For example, while bit-level specifications are typically simple and concise, their word-level implementations are often daunting. Word-level implementations are essential because they can deliver an order of magnitude speedup, which is important for servers where security-related processing can consume up to 95% of processing capacity [20]. Converting bit-level implementations to word level implementations is akin to vectorization, but the characteristics of bit-streaming codes render vectorizing compilers largely ineffective. In fact, widely used cipher implementations often achieve performance thanks to algorithm-specific algebraic insights that are not available to a compiler.

Additionally, correctness in this domain is very important because a buggy cipher may become a major security hole. In 1996, a release of the BlowFish cipher contained a buggy cast (from unsigned to signed characters) which threw away two thirds of the

Group Activity

Short exercise

Draft Your Research Press Release

- What is the problem?
- Who is the person it affects?
- What is the proposed solution?
- How does the solution work?
- What is the Research Plan?
- The steps that take us from now to solving the problem

CRA
Computing Research Association

Summary

Take home ideas

- “The closest path between two points is not a straight line. It is the path you ~~know~~ find”.
- Know yourself.
- Be curious.
- Connect with others.
- Have goals, until you find better ones.
- Be ready to change.

CRA
Computing Research Association