Enabling Polyhedral Optimizations in Julia

Google Summer of Code 2016

Matthias Reisinger

November 3rd, 2016

FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Introduction

- Julia is a programming language for technical computing
- Official language implementation uses LLVM as JIT compiler
- Allows the use of Polly in Julia
- Functions that should be optimized are marked with `@polly`:

```
@polly function foo()
    # your code
end
```

What can be optimized?

- Polyhedral optimization is limited to **Static Control Parts**
- Classical SCoP definition:
  - Static control flow
  - Loop bounds, conditions, and array accesses are affine in parameters and induction variables
  - No side effects
- Goal is to support the subset of Julia that satisfies these requirements

Results

- Experiments based on Julia port of **PolyBench** which contains 30 SCoPs
- Measured both with and without bounds checks

Results

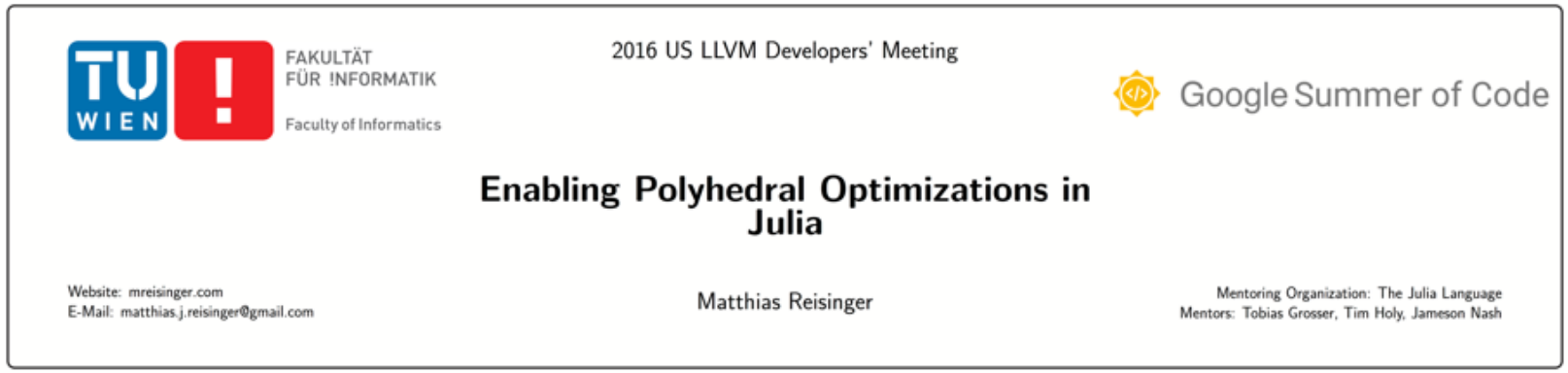
- Experiments based on Julia port of **PolyBench** which contains 30 SCoPs
- Measured both with and without bounds checks
- **Number of SCoPs** detected by Polly could be increased
  - from 19 to 28 when bounds checks are disabled
  - from 0 to 27 when bounds checks are enabled

Results

- Experiments based on Julia port of **PolyBench** which contains 30 SCoPs
- Measured both with and without bounds checks
- **Number of SCoPs** detected by Polly could be increased
  - from 19 to 28 when bounds checks are disabled
  - from 0 to 27 when bounds checks are enabled
- **Speedup** of detected SCoPs (geometric mean):
  - 1.05 when bounds checks are disabled
  - 1.07 when bounds checks are enabled

More infos at...

- My blog http://mreisinger.com
- The poster session



FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

2016 US LLVM Developers' Meeting

Google Summer of Code

Enabling Polyhedral Optimizations in Julia

Website: mreisinger.com
E-Mail: matthias.j.reisinger@gmail.com

Matthias Reisinger

Mentoring Organization: The Julia Language
Mentors: Tobias Grosser, Tim Holy, Jameson Nash

Overview

- This GSoC project enabled the use of polyhedral optimizations for Julia programs based on LLVM's loop optimizer Polly.
- Functions that should be optimized are annotated with `@polly`, i.e. programmers explicitly decide when to use Polly:

```
@polly function foo()
    # your code
end
```

- Polly's bounds check elimination logic was enabled for multidimensional array accesses.

What can be optimized?

- Polyhedral optimization is limited to Static Control Parts (SCoPs).

Background: Bounds Checks

- By default Julia emits bounds checks for each array access.
- For example, an access like `A[i] = 0` would actually translate to:

```
if (i < 1 | i > length(A))
    throw(BoundsError())
end
A[i] = 0
```

- Bounds checks lead to runtime overhead, particularly inside loops.
- They can be disabled with `@inbounds`.

Eliminating Bounds Checks

- Polly is able to eliminate bounds checks or hoist them before loops.
- Enabling these capabilities in Julia required adaptions to the code