P57 - Automatic Annotation Inserter

Massimo Fioravanti

Polimi

June 17, 2019

Abstract

Taffo compiler is a collection of llvm passes used to perform precision tuning. The user is allowed to annotate variables and functions with bounds and properties, so that Taffo can make optimizations that are otherwise impossibile.

```
float lerp(
    float start,
    float end,
    float __attribute(annotate("scalar(range(0, 1))")) i)
{
    return ((1-i) * start) + (i * end);
}
```

The floating variable i will be assumed to be in the range [0,1].

Abstract

These annotations can quickly grow in complexity. When a variable is a struct it is neccessary to specify the annotation of each member of the struct.

```
struct Internal float v1, float v2, float v3;
struct External float u1, struct Internal u2;

struct External __attribute(annotate("struct[
   scalar(...), struct[
      scalar(...),
      scalar(...),
      scalar(...)]]")) var;
```

The objective is to decouple the internal reppresentation of taffo annotations from the one used by end users, allowing that to be simpler.

Solution

Most of the complexity arises from the impossibility of reusing the same struct annotation multiple time, since there is no way to give a name to them.
We wish to be able to express a concept like:

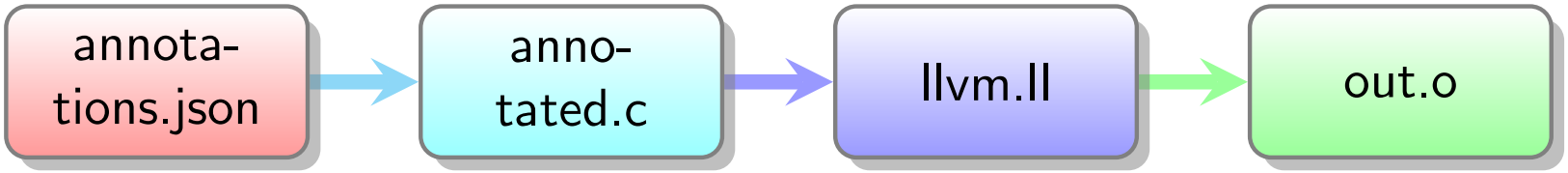
```
{
   "struct": "internal",
   "content": [{...}, {...}, {...}]
},
{
   "struct": "external",
   "content": [{...}, "internal"]
}
```

This allow us to express a hierarchical structure without having to express it with syntax.

Insertion

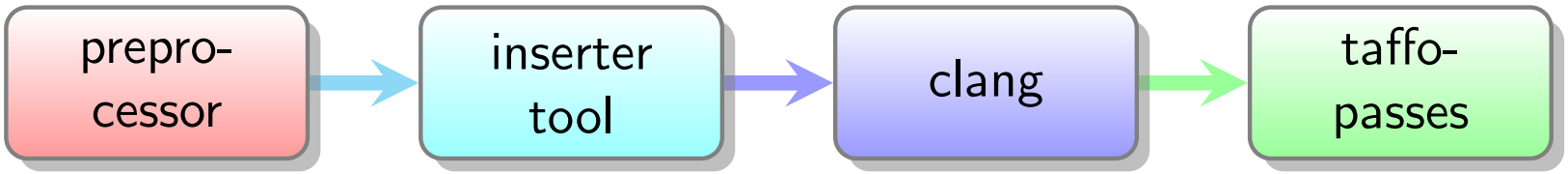
We decided to use a subset of the json language to rappresent our user annotation language. This allow us reuse a well known library to load the user defined annotations from a json file.

Since the user will often tune the annotations we wish to make sure that the user does not need to change the source files every time.



Insertion

Since we only care about declaration we do not need to worry about macros, we can run the preprocessor and save the output in a temporary file, parse the content temporary, locate the declarations insert the annotation, and then compile as normal.



The inserter tool only need to be able to parse c and cpp files. Clang libtooling is the standard solution for this operation.

Clang tools

Libtooling allows us to write a recursive visitor of the clang Abstract Syntax Tree. Every time we encounter a declaration we check if it is a VarDecl of a variable that must be annotated. If it is we can insert the annotation at the before at the beginning. Libtooling includes rewriting utilities.

Many edge cases are present, but most of them are solved by running the preprocessor before the tool. Still other kinds of code generations, such as c++ templates, can elude the insertion.

Results

This pipeline allowed us to annotate the Approximate Computing Benchmarchs with almost no neccessity of insert manually the annotations in the source code, as well as providing a easy to use syntax to the final users that is independent from the internal reppresentation.

Future Works

The user language is small and easy to use and there is no particular need to extend it.

As is always the case with c++ syntax there is plenty of room to cover insertion edge cases, as well as detecting impossible insertion and alert the user that such edge cases are taking place.

To improve build performance it would be useful to integrate the preprocessor and the annotation inserter with tools that provide incremental builds so that it is not neccessary to insert the annotations at each invocation of the compiler but only when the annotation file or the source file are changed.