Code Optimization and Transformation Course Project on:

Bounds Checking Elimination

The Code Optimization and Transformation course exam is composed by two parts. One is an oral test, the other is an homework, to be terminated before course last call. To pass the whole exam, you must get a pass grade from both the test and the homework. The homework must be taken in pairs.

During the lab classes, the LLVM [5] compiler has been introduced. The homework must use the 3.0 release version of LLVM. A sample project – COT passes [6] – is available on GitHub [3]. It must be used as a starting point for the homework. LLVM testing framework [2] must be used to validate the implementation.

Sources must versioned using Git [7]. A good tutorial can be found here [1]. Sources must be published on GitHub [3].

Assignment

High level languages, like Java, often generates code protecting array accesses – i.e. the subscript used for indexing the array is checked before accessing the memory in order to detect whether the access in inside the bounds of the array.

Array accesses in C are not checked. You are required to implements an header-file only library to performs checked array accesses. An example of library usage is show in Figure 1(a).

```
#include "sbounds.h"

void init(int *foo, int n) {
  for(int i = 0; i != n; ++i)
    aset(foo, n, i, 0);
}
```

(a) User code

```
void init(int *foo, int n) {
  for(int i = 0; i != n; ++i) {
    if(i < 0 || i >= n)
      abort();

    foo[i] = 0;
  }
}
```

(b) Unoptimized code

Figure 1: Examples of performing safe array accesses

Functions provided by the library performs bound checking before accessing the memory. If the aset function used in Figure 1(a) is expanded, you get a code like the one reported in Figure 1(b)

The goal of this project is to detect such checks and optimized them – e.g. moving them outside of the loop or removing them when possible. The optimization performing such task is called *Bounds Checking Elimination* [4].

Advices

As stated before, C compilers does not emit bounds checking code because they usually do not known whether a memory access refers to an array – e.g. access arrays through pointers. The array access library you have to develop allows to generate bounds checking code at compile-time. In order to force its functions to be always in-lined by the compiler, mark them with the always_inline attribute.

Bounds checking elimination requires observing how array subscripts are used to access the memory. The LLVM SCEV framework can provide the information you need.

References

[1] Scott Chacon. Pro Git. URL `http://git-scm.com/book`.

[2] John T. Criswell, Daniel Dunbar, Reid Spencer, and Tanya Lattner. LLVM Testing Infrastructure Guide. URL `http://llvm.org/releases/3.0/docs/TestingGuide.html`.

[3] GitHub Inc. GitHub. URL `http://github.com`.

[4] Steven S. Muchnick. *Advanced Compiler Design and Implementation*, chapter Loop Optimizations. Morgan Kaufmann.

[5] University of Illinois at Urbana-Champaign. Low Level Virtual Machine. URL `http://www.llvm.org`.

[6] Ettore Speziale. Compiler Optimization and Transformation Passes. URL `https://github.com/speziale-ettore/COTPasses`.

[7] Linus Torvalds. Git. URL `http://git-scm.com`.