Debugging of optimized code: Extending the lifetime of local variables and parameters

Wolfgang Pieb
October 18, 2017
Motivation

• Local variables and parameters (including the this pointer) are often optimized away soon after the last point of use.

```cpp
void A::func()
{
    if (<last use of this>) {
        handle_error();
    }
}
```

• By artificially extending the lifetime of these locals and parameters through the end of their lexical scopes we make them visible for debugging purposes.
-O3 -g

-attribute__((noinline)) void A::func() {
  if (foo(m_i, m_d)) {
    handle_error();
  }
}

int main()
{
  A a;
  a.func();
}

-O3 -fextend-lifetimes -g

-attribute__((noinline)) void A::func() {
  if (foo(m_i, m_d)) {
    handle_error();
  }
}

int main()
{
  A a;
  a.func();
}
Implementation

• New clang switches -fextend-lifetimes and -fextend-this-ptr
• New llvm intrinsic `llvm.fake.use()`

```c
define i32 @_Z3fooi(i32 %param) {
   ...
   call void (...) @llvm.fake.use(i32 %param)
}
```

• The front-end issues calls to `llvm.fake.use()` for all user-defined local variables and parameters at the end of their respective lexical scopes.

• With -fextend-this-ptr, only the `this` pointer’s lifetime is extended.

• Analogous to generating of end-of-lifetime markers.
Example

extern void used(double);
extern void usei(int);
double globd;
int globi;

void foo(int param)
{
  double d = globd;
  if (param) {
    int j = globi;
    usei(j);
  }
  used(d);
}

define void @foo(i32 %param) … {
  entry:
  %d = load double, double* @globd, align 8
  ...  
  br i1 %tobool, label %if.end, label %if.then
  if.then
  %j = load i32, i32* @globi
  tail call void @usei(i32 %j)
  tail call void (...) @llvm.fake.use(i32 %j)  <= after call to usei()
  br label %if.end
  if.end:
  tail call void @used(double %d)
  tail call void (...) @llvm.fake.use(double %d)  <= after call to used()
  tail call void (...) @llvm.fake.use(i32 %param)  <= end of the function
  ret void
}
Backend implementation

- `llvm.fake.use()` is translated into the new `FAKE_USE` machine op with the intrinsic’s argument as operand.
- `FAKE_USE` is a meta instruction (i.e. does not produce any executable code).
- Some GVN optimizations are suppressed for `FAKE_USE` operands.
- SROA on pointer operands of `FAKE_USE` is disabled.
- Type legalization needed to learn about `FAKE_USE` and its operands.
Effect on debug location information

• Measuring coverage by determining the percentage of code that is covered within a variable’s lexical scope.

- Code range for variable’s parent scope
- Variable’s first DEF
- Location information w/ -O3 -g
- Improvement w/ -O3 -g -fextend-lifetimes

• Game 1: Cumulative coverage improvement by 15%
• Game 2: Cumulative coverage improvement by 14%
Effect on runtime performance

As percentage of execution time

Game 1  Game 2  Bullet  zlib

- O3  -O3 -fextend-lifetimes  -O3 -fextend-this-ptr
Conclusion

- Debugging of optimized code can be improved by extending the lifetime of local variables and parameters artificially.
- The impact on performance is small (5-7%).
- Positive feedback from users.
- The proposed -Og mode (optimize for debugging) could make use of this functionality.