Profile-Guided Optimization in the LDC D compiler

Implementation and Benefits

Kai Nacke

5 February 2017

LLVM dev room @ FOSDEM‘17

What is Profile-Guided Optimization?

- Simple idea: provide more information to enable better optimization
- Instrument source with counters
  - Count function calls
  - Count branches taken / not taken
  - Count how often function **`A`** call function **`B`**
- Create profile from run of instrumented binary
- Result depends on quality of profile
  - Can harm performance if profile does not fit use case
- No silver bullet – be sure to measure!

What is this D language?

- System programming language
- C-like syntax
- Static typing, modules
- Polymorphism, functional style, generics, …
- Template meta programming
- Automatic memory management
- Compile time function execution

LDC combines the DDMD reference frontend with the power of LLVM

Implementation in the LDC compiler

The following steps were required to implement PGO in LDC

Implementation by Johan Engelen!
Thank you!!!

- Source instrumentation
  - Inserts calls to LLVM intrinsics
  - Typical pattern

```
auto &PGO = irs->func()->pgo;
PGO.setCurrentStmt(stmt);
...
PGO.emitCounterIncrement(stmt);
```

Implementation in the LDC compiler

- New optimization pass InstrProfilingPass

```
#if LDC_LLVM_VER >= 309
    mpm.add(createInstrProfilingLegacyPass(options));
#else
    mpm.add(createInstrProfilingPass(options));
#endif
```

- Functions for reading and writing profile data

- Integration of LLVM profile runtime library
  - Driver needs to know this library
  - profile-rt is added

How to use Profile-Guided Optimization

- Latest release of LDC has PGO support enabled by default
- Follow these steps
  1. Compile with instrumentation turned on
     `ldc2 -fprofile-instr-generate pgotest.d -of=pgotestinstr`
  2. Run binary to produce raw profile **`default.profraw`**
     `./pgotestinstr`
  3. Convert the raw profile
     `ldc-profdata merge default.profraw -o=pgotest.profdata`
  4. Compile with profile data
     `ldc2 -fprofile-instr-use=pgotest.profdata pgotest.d`
- Measure!

Indirect Call Promotion (ICP)

LLVM 3.9 or later!

- Indirect calls occur very often in OO languages (virtual functions)
- If a ‘likely’ called function is known then an optimization is to check for the ‘likely’ function and call this function directly
- PGO helps to find the ‘likely’ called function
- Enables further optimization, e.g. function inlining

Example: ICP at D level (manually)

```
int icp() {
    return 42;
}

int function() fptr = &icp;

int main() {
    int a = 0;
    foreach (i; 0..1000) {
        a += fptr();
    }
    return a;
}
```

```
int icp() {
    return 42;
}

int function() fptr = &icp;

int main() {
    int a = 0;
    foreach (i; 0..1000) {
        if (fptr == &icp)
            a += icp();
        else
            a += fptr();
    }
    return a;
}
```

Example: ICP at D level (manually)

```
auto is_likely(alias Likely, Fptr, Args...)(Fptr fptr, Args args) {
    return (fptr == &Likely) ? Likely(args) : fptr(args);
}

int icp() {
    return 42;
}

int function() fptr = &icp;

int main() {
    int a = 0;
    foreach (i; 0..1000) {
       a += fptr.is_likely!icp();
    }
    return a;
}
```

Example: ICP at IR level - original

```
forbody:
  ...
  %1 = load i32 ()*, i32 ()** @_D7pgotest4fptrPFZi, align 8
  %2 = tail call i32 %1()
  ...
```

Example: ICP at IR level - instrumented

```
forbody:
  call void @llvm.instrprof.increment(i8* getelementptr inbounds ([6 x i8],
             [6 x i8]* @__profn__Dmain, i32 0, i32 0), i64 4, i32 2, i32 1)
  ...
  %5 = load i32 ()*, i32 ()** @_D7pgotest4fptrPFZi
  %6 = ptrtoint i32 ()* %5 to i64
  call void @llvm.instrprof.value.profile(i8* getelementptr inbounds (
             [6 x i8], [6 x i8]* @__profn__Dmain, i32 0, i32 0), i64 4,
             i64 %6, i32 0, i32 0)
  %7 = call i32 %5()
  ...
```

Example: ICP at IR level – profile data

```
> ldc-profdata show --all-functions default.profraw
Counters:
  _D7pgotest3icpFZi:
    Hash: 0x0000000000000000
    Counters: 1
    Function count: 1000
  _Dmain:
    Hash: 0x0000000000000004
    Counters: 2
    Function count: 1
Functions shown: 2
Total functions: 2
Maximum function count: 1000
Maximum internal block count: 1000
```

Example: ICP at IR level – PGO applied

```
forbody:
  ...
  %1 = load i32 ()*, i32 ()** @_D7pgotest4fptrPFZi, align 8
  %2 = icmp eq i32 ()* %1, @_D7pgotest3ictFZi
  br i1 %2, label %if.end.icp, label %if.false.orig_indirect, !prof !32

if.false.orig_indirect:
  %3 = tail call i32 %1()
  br label %if.end.icp

if.end.icp:
  %4 = phi i32 [ %3, %if.false.orig_indirect ], [ 42, %forbody ]
  ...

!32 = !{!"branch_weights", i32 1000, i32 0}
```

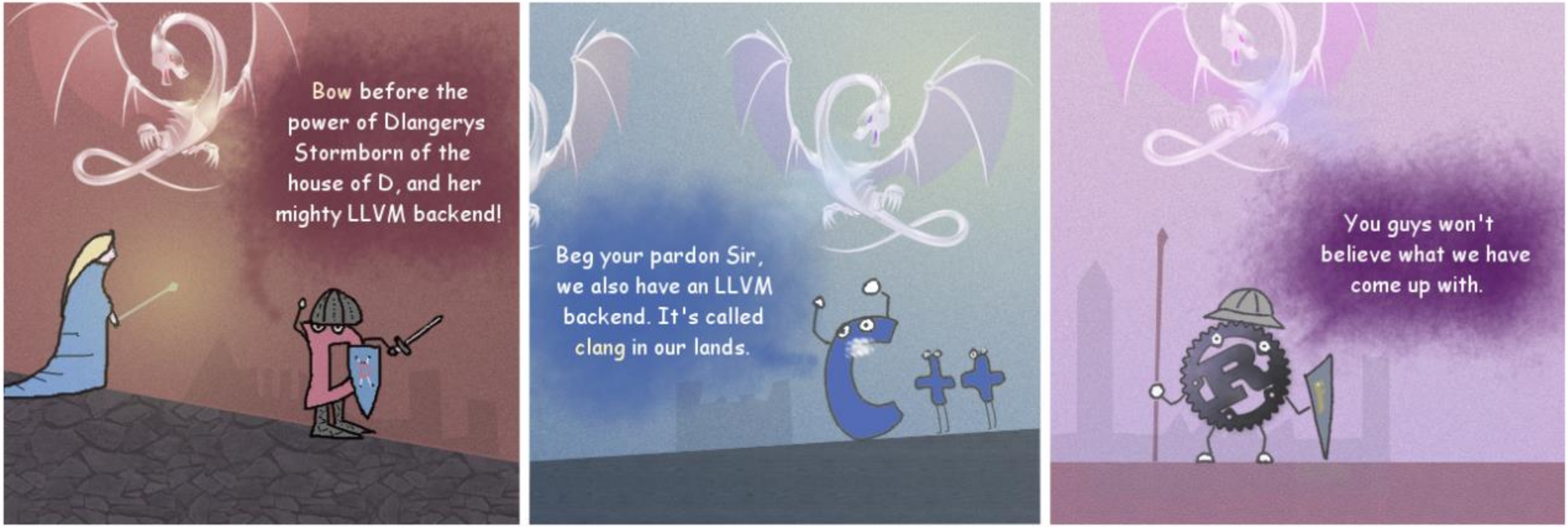
Benefits

- D code of LDC becomes faster by 7% on real test case
  - LDC is a mix of D and C++ code. PGO was applied to D code only.

- Possible enhancement: Virtual Call Promotion (VCP)
  - Apply ICP to **`vtable`** based virtual function calls
  - Only sample implementation

- It’s worth to have PGO in your optimization toolbox!

Questions?



Resources

About PGO in LDC

- Profile-Guided Optimization with LDC
- PGO: Optimizing D's virtual function calls
- LDC LLVM profiling instrumentation

About PGO in LLVM

- Profile-based Indirect Call Promotion
- PGO in LLVM: Status and Current Work

Resources

About D and LDC

- LDC
- D