Kai Nacke
DConf 2016

Everywhere!
Agenda

• Introduction
• LDC internals
• Porting and extending druntime
• Porting and optimizing Phobos
• Testing with Continuous Integration
Introduction

• D is a systems programming language

• Should run everywhere
  – Needs at least a 32bit CPU

• Reference implementation targets x86/x86_64 based systems
Introduction (2)

• There are more than x86 based devices
  – Tablets and smartphones are mostly ARM-based devices
  – IoT devices use AVR, ARM, MIPS or other CPUs
  – Servers use POWER, SPARC or AArch64 CPUs
  – ....

Non-x86 targets are required for success of D
LDC: Supported targets

- x86
- x86_64
- ARM
- PPC

Work in progress:

- AArch64
- MIPS64
- SystemZ
- ARM
- ARM
- x86_64
LDC: Not yet supported targets

• Partial list of not yet supported targets:
  – Any SPARC based operating system (32/64 bit)
  – OpenBSD, DragonFly
  – FreeBSD and NetBSD on non-Intel hardware
  – GPUs (NVIDIA, AMD)
  – AIX

• ... and many more
What is the challenge?

• Very x86/x86_64 centric view in druntime and Phobos because of development history

• Example: Use of x86 assembler in test cases

```d
void crash(int x)
{
    if (x==200) return;
    asm { int 3; }
}
```

```d
void crash(int x)
{
    if (x==200) return;
    import ldc.intrinsics;
    llvm_debugtrap();
}
```
LLVM challenges

• Not all targets are feature-complete

• Typical areas which can require improvement
  – TLS
  – Exception handling

• clang is often the only client
  – Using LDC can discover some hidden bugs
Current way of porting

• We treat the last C++ version of LDC as a version with long-term support

• Steps for porting to a new platform:
  – Compile or install LLVM and libconfig
  – Compile LDC
  – Fix all compile errors and test suite failures
  – Create pull request
LDC internals

- LDC driver may need some tweaks
  - Check definition of D version identifier

```c
static void registerPredefinedTargetVersions() {
    switch (global.params.targetTriple.getArch()) {
        case llvm::Triple::RISCV:
            VersionCondition::addPredefinedGlobalIdent("RISCV");
            break;
    ....
```

With **ldc -v** you can check which D version identifiers are defined
LDC internals (2)

- LLVM requires compilers to implement ABI (see `gen/abi*.*cpp`)

Wrong or missing ABI implementation causes failures in test suite
Extending druntime

• Adding basic support for an OS requires
  – Extending the POSIX modules if needed
  – Adding OS specific modules

• Examples: NetBSD, Android, iOS

In general: If you miss a crucial part then you will get compile or linker errors
Porting druntime

• Adding a new CPU architecture requires more effort

  – Add assembly code to `core.thread`

```d
private void callWithStackShell(void delegate(void* sp))
{
    version (LDC)
    {
        version (RISCV)
        {
            import ldc.llvmasm;

            size_t[1] regs = void;
            __asm(`sd $$16, 0($0)`, "r", regs.ptr);
        }
    }
    ....
}
```
Porting druntime (2)

• Add assembly code to `threadasm.S`
  – Not required if `ucontext_t` is supported
• Check 128bit CAS support in `core.atomic`
  – Simply run the unit test
• Implement `core.stdci.stdarg` if needed

Be aware if used: `core.cpuid` only supports x86/x86_64
I am working on an auxv-based solution for Linux
Section support in druntime

• Sections determine location of code and data
• Example: ELF

• Crucial for GC support and shared libraries
• DMD solution not applicable for LDC
Section support in druntime (2)

- Porting effort varies: ELF not used everywhere
- Even differences if ELF is used
  - Different offsets for TLS section
  - zLinux does not have `__tls_get_addr()`
  - LDC does not support shared libraries on Solaris

Symptoms if there is an issue:

- Unit tests allocating a lot of memory fail in unpredictable ways
- Failure goes away if linked against stub GC
Floating point support

• Complete support only for 32bit, 64bit and 80bit reals
• 128bit IEEE quadruple and IBM extended doubledouble formats have only partial support

Unit tests of `c.i.convert` and `c.i.hash` do not compile for these types
Porting Phobos

• Phobos builds on druntime and requires usually less changes

• Most changes are due to use of system specific modules

You get a compile error if you need to add Code. Example: A missing import
Phobos and floating point

- **std.math** is very x87 FPU centric
  - A lot of inline assembly
  - Accuracy of unit tests tuned for 80bit reals
- **Struct IeeeFlags** must be implemented
- 128bit floating point formats only partial implemented

**std.math** causes a lot of trouble if you have an incomplete supported float format.
Optimizing Phobos

• Re-Implement D module `s.i.m.biguintnoasm` with assembly (partially done for ARM)

• Explore efficient implementations of digest algorithms
  – POWER8 and AArch64 have special instructions
  – MIPS OCTEON has crypto co-processor

• Research other possible optimizations!
Typical Porting Trap

• The D spec says:

The \texttt{extern (C)} and \texttt{extern (D)} calling convention matches the C calling convention used by the supported C compiler on the host system.

• Does not mention non-POD structs!
• NRVO implies that all non-POD structs are passed in memory

CTFE test cases (and others) can fail if you do not pay attention to the ABI
Future way of porting

- DMD frontend is now written in D
- Preferred way of porting is cross-compiling
- Cross-compiling requires floating point support independent of host (pending PR #5471)

Currently you can get wrong results wrt. floatings points if you use cross-compiling
Continuous Integration

• Tests are very important for a compiler

• The LDC developers use Continuous Integration
  – Test suite executed for each commit / PR

• Different CI servers for x86_64
  – Travis CI: Linux and OS X, LLVM 3.5-3.9
  – CircleCI: Linux with LLVM 3.9
  – AppVeyor: Windows
Continuous Integration (2)

- buildbot is used for ARM and OpenPOWER
- More builders can be added
Let's port LDC!
Reference: Images

By Larry Ewing, Wikimedia Commons

By Google, Wikimedia Commons

By Apple, Wikimedia Commons

By Anton Gural, Wikimedia Commons

By Microsoft, Wikimedia Commons
Reference: Images

Icon for Solaris, [Wikimedia Commons](https://commons.wikimedia.org/)

ARM [Wikimedia Commons](https://commons.wikimedia.org/)

OpenPOWER[™] ChrispyD1080, [Wikimedia Commons](https://commons.wikimedia.org/)

SPARC Sun Sparc, [Wikimedia Commons](https://commons.wikimedia.org/)

x86 logo, [Wikimedia Commons](https://commons.wikimedia.org/)
Reference: Images

Exclamation marks, [Wikimedia Commons](https://commons.wikimedia.org)

NetBSD logo, [Wikimedia Commons](https://commons.wikimedia.org)

OpenBSD logo, [Wikimedia Commons](https://commons.wikimedia.org)

OS X El Capitan logo, [Wikimedia Commons](https://commons.wikimedia.org)

DragonFLyBSD logo, [Wikipedia](https://en.wikipedia.org)
Reference: Images

RISC OS logo, [Wikipedia](https://en.wikipedia.org/wiki/RISC_OS)

ELF layout, [Wikimedia Commons](https://commons.wikimedia.org/wiki/File:ELF_Format.svg)

LLVM logo by Apple, [LLVM.org](https://llvm.org)