Connecting the EDG front-end to LLVM

Renato Golin, Evzen Muller,

Jim MacArthur, Al Grant

ARM Ltd.

Outline

- Why EDG
- Producing IR
- ARM support

EDG Front-End

- LLVM already has two good C++ front-ends, why add yet another one?
  - We already use EDG in our compiler and would like to know how LLVM would support all EDG features and extensions (plus our modifications), at both IR and CodeGen levels
  - EDG supports the complete set of C/C++ standards (all versions), plus many features of the new standard (C++0x)
  - It's feature stable; newer versions of the compiler should still compile old code the same way
    - Including extensions, platform choices
    - Consistent error reports (especially with warnings as errors)
  - Some ARM customers can’t use GCC (GPLv3) and Clang is still immature

EDG Front-End

- The benefits for the LLVM community are:
  - Cross-checking IR/code generation against a different front-end
    - It's a fresh set of eyes on the same problems
    - We found some CodeGen bugs (some patches sent)
    - We found problems in structures, unions, bitfields
    - As well as NEON, Exception Handling, Run-Time ABI
  - We tested using vectors instead of structures for NEON
    - So far, it's generating good code
  - Extend IR/code generation support for the additional features, mainly EABI
  - Some EDG users are asking about the bridge...

EDG + LLVM

- Why EDG
- **Producing IR**
- ARM support

General Impressions

- LLVM is a great platform to work on
  - It's easy to generate IR / Metadata
    - Good IR validation infrastructure
    - Asserts get most of the consistency problems
  - Coding standard is well thought of and modern
    - Makes good use of C++ capabilities
    - Consistent, frequently refactored
    - Has good ways to debug (`dump`, `viewGraph`)
- Generated code quality is good
  - Good support for ARM architectures

Type System

- Sign information
  - EDG has it on type, easier for front-end development
  - LLVM has it on instructions, easier for code-generation
  - We need to keep track to do casts, widening...
    - Ex: `sext, zext, uitofp, sitofp`
  - ...comparison functions
    - Ex: `icmp uge` vs. `icmp sge`

Type System

- Default alignment in EDG:
  - Type and variable alignment
  - Variable overrides type alignment
  - Structure and Union alignment
- Alignment in LLVM on instructions, globals, allocas
  - Default alignment comes from Data Layout
  - Special cases (structures, unions, bitfields) have to be carefully hand-crafted
  - Still, alignment in instructions are necessary to make bitfields work properly
  - Throw away EDG info and partially reconstruct later

Type System

- Automatic C casting
  - EDG sometimes relies on C's implicit conversions
    - Mostly on type promotions
  - LLVM doesn't accept any implicit cast
  - Adding non trivial helpers to avoid bloating the IR
    - Fiddling with types and sizes to determine the expected implicit cast
    - Caused many LLVM failures by being too permissive or getting the cast wrong
    - Took us a while to be auto-cast free, but it was worth it

C++ support

- C++ support was fairly simple
  - EDG lowers C++ into C, so most support was already present
- Easier parts were
  - Classes became structures, virtual functions became calls to pointers in an array
  - Static construction/destruction identical to LLVM
- Bigger problems were
  - Exception Handling (no EHABI)
  - EDG storage class doesn't map directly to LLVM linkage type (some magic was required)
  - Explicit allocation of parameters (thunks and other)

Structures

- Argument passing
  - Code generated for Struct ByVal is not EABI compliant
  - Missing feature in ARM's backend for structures
  - Following Clang's example, converting structures to arrays, which gets correctly lowered as ByVal
- Return Value
  - Indirect return has the same problem, as it becomes the first argument

Unions

- Unions are target-dependent from the front-end
  - Lots of flames, but most agree it could be better
  - The number of work-arounds to make it conform to the C++ standard is quite big
    - Create N structure types (for N initialisers)
    - Cast to i8 arrays, memcpy, and back
    - Keep track of alignment when doing so
  - Static initialization and alignment pose the worst problems
- Implementing it in the back-end might require several changes
  - All targets must change simultaneously

Bitfields

- Zero-sized anonymous bitfields
  - In the C standard, bitfields finish the packing of the previous field
  - Armcc also uses it for general structure alignment
  - But this cannot be easily represented in LLVM IR
  - For example:

```
struct A {
  int :0;
  char a;
} S;
```

compiler	GCC x86	Clang ARM	Armcc	GCC ARM
sizeof(S)	1	1	4	4

  - Maybe consider introducing i0 integer type?

Exception Handling

- ARM EHABI exception handling has not (yet) been implemented
  - Plans to support it in MC in the near future
- DwarfException.cpp cannot be changed nicely to support EHABI; Any change will be considerably invasive
  - We have no ELF writer for ARM
  - Codesourcery's GAS doesn't understand the tables
  - Need to export it to different ELF sections
  - Need to support Generic and Compact EHABI
- There are solutions
  - Implement EHABI in MC
  - Write Dwarf exceptions directly to ELF

NEON

- NEON instructions are fully(?) represented in TableGen
  - Some instructions can be represented via pattern-matching, others via intrinsics:
    - `add(zext, zext) = VADDL.U*`
    - `@llvm.arm.neon.vhadds.*() = VHADD.S*`
    - `@...vabds.*() + add = VABA.S*`
- `arm_neon.h` in Clang reflects those choices
  - But neither ARM's nor GCC's `arm_neon.h` do
- Structures vs. Vectors in NEON type representation
  - Source compatibility is important, but the front-end can recognize NEON structures and transform them into vectors in IR (we do that)

Suggestions

- Add target attributes in IR header
  - Some generic enough to be used by any back-end
    - Could the union type benefit from this?
  - Some target specific
    - Like build attributes for ARM
- Add validation to the debug data in IR
  - MDNode* is too opaque for validation
  - Helper classes are too simple, but could do basic validation
- Unions? Bitfields?
- EABI Struct ByVal?

Architecture Support

- Why EDG
- Producing IR
- **ARM support**

Architectures supported

- We tested Dhrystone on all supported architectures
  - v4 to v7 (A, R, M)
  - ARM, Thumb
  - Soft and Hard FP, NEON
  - Specific instructions (like SMUL)
- Our internal tests run mainly on 7TDMI and Cortex-A8
- NEON tests in early stages
  - Almost all intrinsics get compiled to NEON instructions correctly
  - We reported some errors in bugzilla

Still missing

- EABI support
  - We've added some RT-ABI (FP, REM and Memset)
  - But there are still things to do (Ex. EHABI, AAELF)
- MC support
  - Refactor ARM MC to be less ASM-focused
  - ARM ELF
  - Exception Handling
  - ARM ASM dialect?

Questions?