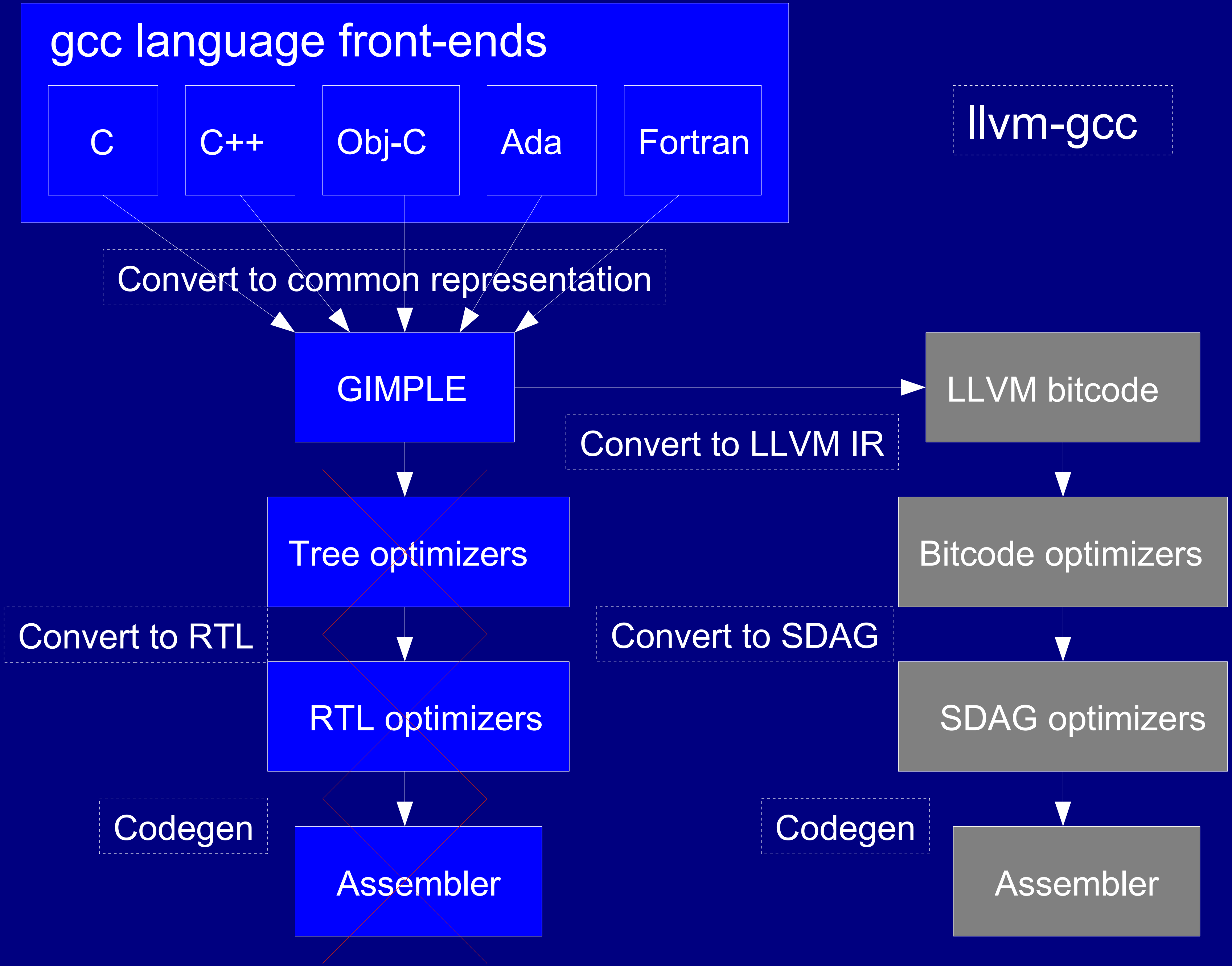
Reimplementing llvm-gcc as a gcc plugin

Duncan Sands

Deep Blue Capital

What is llvm-gcc?

- A drop-in replacement for gcc
- Modified version of Apple's gcc-4.2.1
  - gcc optimizers replaced with LLVM optimizers
  - gcc codegen replaced with LLVM codegen
  - About 300 files changed (excluding testsuite)
- Licensed under the GPL version 2 or later


gcc language front-ends
C
C++
Obj-C
Ada
Fortran
llvm-gcc
Convert to common representation
GIMPLE
LLVM bitcode
Convert to LLVM IR
Tree optimizers
Bitcode optimizers
Convert to RTL
Convert to SDAG
RTL optimizers
SDAG optimizers
Codegen
Codegen
Assembler
Assembler

OK, so what is gcc?

- Open source C, C++, Obj-C, Obj-C++, Ada, Fortran and Java compiler from the GNU project
- Version 4.5 maybe late 2010?
- About 3 years worth of changes since gcc-4.2
  - Many bugs fixed
  - Support for the C++0x standard
  - Major libstdc++ improvements
  - Improved support for the Ada 2005 standard
  - Support for the Fortran 2003/2008 standards
  - OpenMP version 3

What is the plugin?

- A shared library, llvm.so
- Loaded by gcc-4.5 at runtime
- Makes gcc-4.5 work like llvm-gcc, using the new gcc plugin architecture

The plugin in action

```
$ gcc hello.c -S -O1 -o -
        .file    "hello.c"
        .section .rodata.str1.1,"aMS",@progbits,1
.LC0:
        .string  "Hello world!"
        .text
.globl main
        .type    main, @function
main:
        subq     $8, %rsp
        movl     $.LC0, %edi
        call     puts
        movl     $0, %eax
        addq     $8, %rsp
        ret
        .size    main, .-main
        .ident   "GCC: (GNU) 4.5.0 20090928 (experimental)"
        .section .note.GNU-stack,"",@progbits
```

The plugin in action

```
$ gcc hello.c -S -O1 -o -  -fplugin=./llvm.so
            .file       "hello.c"
# Start of file scope inline assembly
            .ident      "GCC: (GNU) 4.5.0 20090928 (experimental) LLVM: 82450:82981"
# End of file scope inline assembly


            .text
            .align      16
            .globl      main
            .type       main,@function
main:
            subq        $8, %rsp
            movl        $.L.str, %edi
            call        puts
            xorl        %eax, %eax
            addq        $8, %rsp
            ret
            .size       main, .-main
            .type       .L.str,@object
```

The plugin in action

```
$ gcc hello.c -S -O1 -o -  -fplugin=./llvm.so -fplugin-arg-llvm-emit-ir
; ModuleID = 'hello.c'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-
f64:64:64-v64:64:64-v128:128:128-a0:0:64-s0:64:64-f80:128:128"
target triple = "x86_64-unknown-linux-gnu"

module asm "\09.ident\09\22GCC: (GNU) 4.5.0 20090928 (experimental) LLVM:
82450:82981\22"

@.str = private constant [13 x i8] c"Hello world!\00", align 1 ; <[13 x i8]*> [#uses=1]

define i32 @main() nounwind {
entry:
  %0 = tail call i32 @puts(i8* getelementptr inbounds ([13 x i8]* @.str, i64 0, i64 0))
nounwind ; <i32> [#uses=0]
  ret i32 0
}

declare i32 @puts(i8* nocapture) nounwind
```

Current status

- Immature (project < 2 months old)
- Can compile a lot of C (eg: gcc)
- Can compile some C++ and Fortran
- Can compile a little Ada
- Does not produce debug info
- Does not support exception handling
- Only X86 (32 and 64 bit) for the moment
- Only Linux and Darwin for the moment

Benchmarks

Sqlite3 at -O3 on X86-64 Linux

Compiler	Time to run	Time to compile	Compiler mem use	Executable size	Bitcode size
plugin	5.3 secs	8.8 secs	138 MB	482 KB	1.1 MB
gcc-4.5	4.6 secs	15.5 secs	323 MB	551 KB	N/A
llvm-gcc	5.3 secs	6.4 secs	118 MB	500 KB	1.0 MB
clang	5.4 secs	6.3 secs	81 MB	503 KB	1.6 MB

Benchmarks

Sqlite3 at -O2 on X86-64 Linux

Compiler	Time to run	Time to compile	Compiler mem use	Executable size	Bitcode size
plugin	5.5 secs	7.3 secs	129 MB	405 KB	0.8 MB
gcc-4.5	5.1 secs	9.6 secs	217 MB	423 KB	N/A
llvm-gcc	5.5 secs	5.0 secs	107 MB	413 KB	0.7 MB
clang	5.6 secs	4.9 secs	80 MB	419 KB	1.2 MB

Advantages wrt llvm-gcc

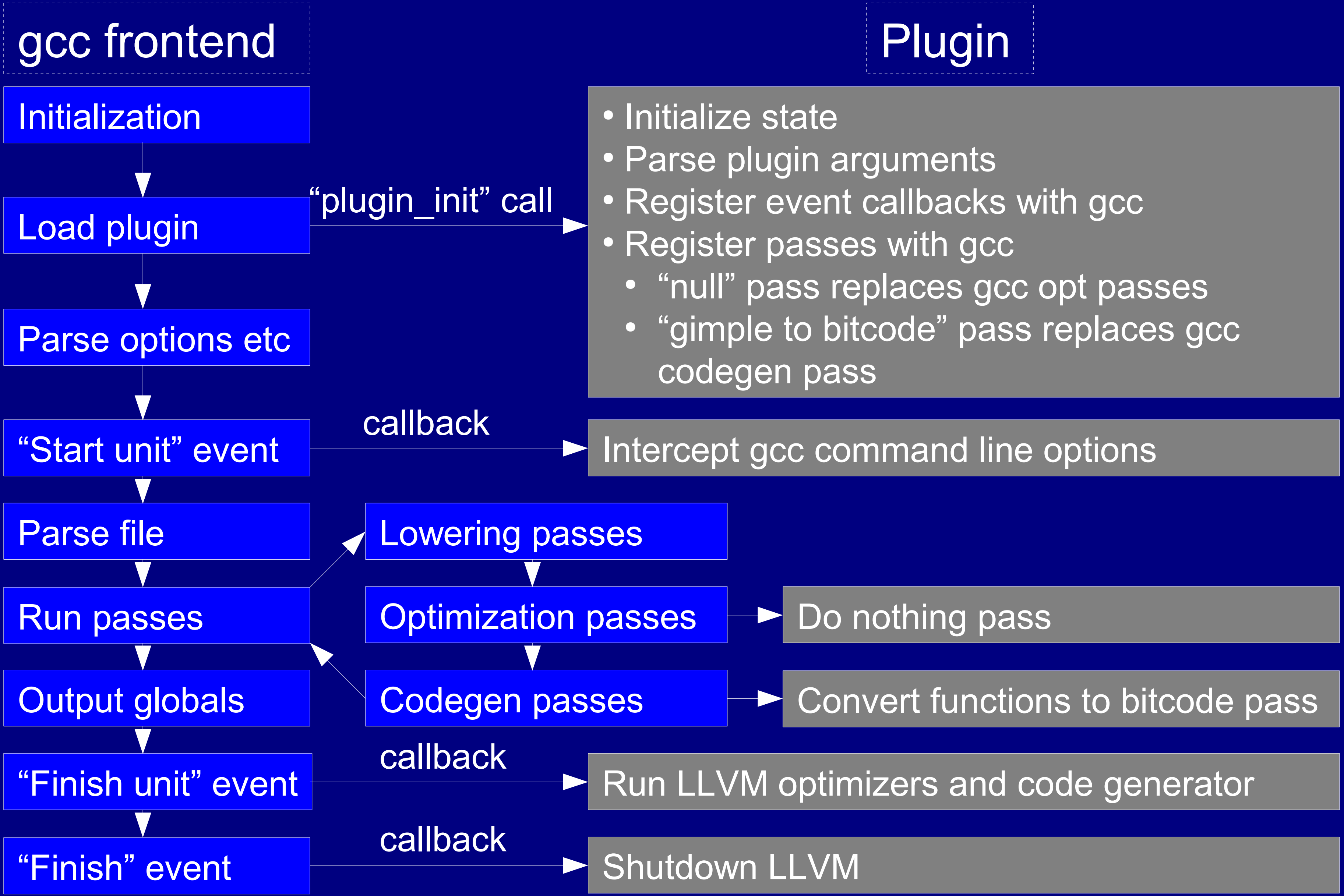
- Benefit from latest gcc frontends
- Easy to try (just add -fplugin=.../llvm.so)
  - Works with unmodified gcc*
- Easy to distribute (one file: llvm.so)
- Compare latest gcc / llvm optimizers
- Plugin quick to build (faster development)

* Currently not true: a minor gcc patch is required

Disadvantages wrt llvm-gcc

- Less mature – may well crash
- Compiles slower and uses more memory
- No Apple extensions, eg no “blocks”
- Clunky incompatible command line flags
  -fplugin-arg-llvm-emit-ir vs -emit-llvm
- Licensed under the GPL version 3

How it works


gcc frontend
Plugin
Initialization
Load plugin
"plugin_init" call
Initialize state
Parse plugin arguments
Register event callbacks with gcc
Register passes with gcc
"null" pass replaces gcc opt passes
"gimple to bitcode" pass replaces gcc codegen pass
Parse options etc
"Start unit" event
callback
Intercept gcc command line options
Parse file
Lowering passes
Run passes
Optimization passes
Do nothing pass
Output globals
Codegen passes
Convert functions to bitcode pass
"Finish unit" event
callback
Run LLVM optimizers and code generator
"Finish" event
callback
Shutdown LLVM

Major issues porting to gcc 4.5

- Gimple in SSA form
  - Need to handle “ssa names” and phi nodes
- Gimple “tuples” data structure
  - New gcc internal representation

SSA form

```
int f(int x, int y, int b) { return b ? x : y; }
```

llvm-gcc

```
...
 %1 = load i32* %b_addr, align 4                  ; <i32> [#uses=1]
  %2 = icmp ne i32 %1, 0                          ; <i1> [#uses=1]
  br i1 %2, label %bb, label %bb1

bb:                                       ; preds = %entry
  %3 = load i32* %x_addr, align 4                 ; <i32> [#uses=1]
  store i32 %3, i32* %iftmp.0, align 4
  br label %bb2

bb1:                                      ; preds = %entry
  %4 = load i32* %y_addr, align 4                 ; <i32> [#uses=1]
  store i32 %4, i32* %iftmp.0, align 4
  br label %bb2

bb2:                                      ; preds = %bb1, %bb
  %5 = load i32* %iftmp.0, align 4                ; <i32> [#uses=1]
...
```

SSA form

`int f(int x, int y, int b) { return b ? x : y; }`

plugin

```
...
  %0 = icmp ne i32 %b1, 0                          ; <i1> [#uses=1]
  br i1 %0, label %"<bb 3>", label %"<bb 4>"

"<bb 3>":                                    ; preds = %"<bb 2>"
  br label %"<bb 5>"

"<bb 4>":                                    ; preds = %"<bb 2>"
  br label %"<bb 5>"

"<bb 5>":                                    ; preds = %"<bb 4>", %"<bb 3>"
  %1 = phi i32 [ %y3, %"<bb 4>" ], [ %x2, %"<bb 3>" ] ; <i32> [#uses=1]
...
```

Tuples representation

z = x + y

llvm-gcc

```
<modify_expr 0x7ffff7e7d3c0
   type <integer_type ...
   side-effects
   arg 0 <var_decl 0x7ffff7e97b40 z type <integer_type ...
   arg 1 <plus_expr 0x7ffff7e7d370 type <integer_type ...
       arg 0 <parm_decl 0x7ffff7f656e0 x type <integer_type ...
       arg 1 <parm_decl 0x7ffff7f65790 y type <integer_type ...
```

plugin

```
gimple_assign <plus_expr, z, x, y>
```

```
svn co http://llvm.org/svn/llvm-project/gcc-plugin/trunk gcc-plugin
```