Workshop on Essential Abstractions in GCC

GCC Control Flow and Plugins

GCC Resource Center
(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

1 July 2012
Outline

- Motivation
- Plugins in GCC
- GCC Control Flow
- Link time optimization in GCC
- Conclusions
Part 1

Motivation
Module Binding Mechanisms

• The need for adding, removing, and maintaining modules relatively independently

• The mechanism for supporting this is called by many names:
  ▶ Plugin, hook, callback, . . .
  ▶ Sometimes it remains unnamed (eg. compilers in gcc driver)

• It may involve
  ▶ Minor changes in the main source
    Requires static linking

  ▶ No changes in the main source
    Requires dynamic linking
Module Binding Mechanisms

- The need for adding, removing, and maintaining modules relatively independently

- The mechanism for supporting this is called by many names:
  - Plugin, hook, callback, ...
  - Sometimes it remains unnamed (e.g., compilers in gcc driver)

- It may involve
  - Minor changes in the main source
    Requires static linking
    \( \text{We call this a } \text{static plugin} \)
  - No changes in the main source
    Requires dynamic linking
    \( \text{We call this a } \text{dynamic plugin} \)
• We view plugin at a more general level than the conventional view
  Adjectives “static” and “dynamic” create a good contrast

• Most often a plugin in a C based software is a data structure containing
  function pointers and other related information
Static Vs. Dynamic Plugins

• Static plugin requires static linking
  ▶ Changes required in gcc/Makefile.in, some header and source files
  ▶ At least cc1 may have to be rebuild
    All files that include the changed headers will have to be recompiled

• Dynamic plugin uses dynamic linking
  ▶ Supported on platforms that support -ldl -rdynamic
  ▶ Loaded using dlopen and invoked at pre-determined locations in the compilation process
  ▶ Command line option
    -fplugin=/path/to/name.so
    Arguments required can be supplied as name-value pairs
Static Plugins in the GCC Driver

Source Program

gcc

Target Program

cc1
cpp
as
ld
glibc/newlib

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Static Plugins in the GCC Driver

Source Program

\[ \text{gcc} \]

Target Program

\[ \text{cc1} \leftrightarrow \text{cpp} \]

\[ \text{as} \]

\[ \text{ld} \]

\[ \text{glibc/newlib} \]

Plugin for a translator in the driver gcc
Static Plugins in the Generated Compiler

- Input Language
- Target Name

Compiler Generation Framework

- Language Specific Code
- Language and Machine Independent Generic Code
- Machine Dependent Generator Code
- Machine Descriptions

Parser
Gimplifier
Tree SSA Optimizer
RTL Generator
Optimizer
Code Generator

Generated Compiler (cc1)

Source Program
Assembly Program

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Static Plugins in the Generated Compiler

Input Language

Compiler Generation Framework

Language and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Plugin for a language front end in cc1

Gimplifier

Tree SSA Optimizer

RTL Generator

Optimizer

Code Generator

Source Program

Generated Compiler (cc1)

Assembly Program

Languages: Specific Code

Languages and Machine Independent Generic Code

Machine Dependent Generator Code

Machine Descriptions

Input Language

Target Name
Static Plugins in the Generated Compiler

Compiler Generation Framework

- Input Language
- Target Name

Plugin for a language front end in cc1

Plugin for adding passes in cc1

Generated Compiler (cc1)

Source Program

Assembly Program

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Static Plugins in the Generated Compiler

- **Input Language**
- **Target Name**

**Compiler Generation Framework**

- **Parser**
- **Gimplifier**
- **Tree SSA**
- **Optimizer**
- **Code Generator**
- **RTL Generator**
- **Optimzer**

**Generated Compiler (cc1)**

- **Source Program**
- **Assembly Program**

**Plugins**

- **Plugin for a language front end in cc1**
- **Plugin for adding passes in cc1**
- **Plugin for code generator in cc1**

**Essential Abstractions in GCC GCC Resource Center, IIT Bombay**
Part 2

**Static Plugins in GCC**
## GCC’s Solution

<table>
<thead>
<tr>
<th>Plugin</th>
<th>Implementation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translator in gcc</td>
<td>Array of C structures</td>
<td>Development</td>
</tr>
<tr>
<td>Front end in cc1</td>
<td>C structure</td>
<td>Build time</td>
</tr>
<tr>
<td>Passes in cc1</td>
<td>Linked list of C structures</td>
<td>Development</td>
</tr>
<tr>
<td>Back end in cc1</td>
<td>Arrays of structures</td>
<td>Build time</td>
</tr>
</tbody>
</table>
struct compiler
{
    const char *suffix;  /* Use this compiler for input files whose names end in this suffix. */
    const char *spec;    /* To use this compiler, run this spec. */
    const char *cpp_spec; /* If non-NULL, substitute this spec for ‘%C’, rather than the usual cpp_spec. */
    const int combinable; /* If nonzero, compiler can deal with multiple source files at once (IMA). */
    const int needs_preprocessing; /* If nonzero, source files need to be run through a preprocessor. */
};
Default Specs in the Plugin Data Structure in \texttt{gcc.c}

All entries of Objective C/C++ and some entries of Fortran removed.

```c
static const struct compiler default_compilers[] = {
    {".cc", "#C++", 0, 0, 0},
    {".cxx", "#C++", 0, 0, 0},
    {".cpp", "#C++", 0, 0, 0},
    {".cpp", "#C++", 0, 0, 0},
    {".c++", "#C++", 0, 0, 0},
    {".C", "#C++", 0, 0, 0},
    {".CPP", "#C++", 0, 0, 0},
    {".ii", "#C++", 0, 0, 0},
    {".ads", "#Ada", 0, 0, 0},
    {".F", "#Fortran", 0, 0, 0},
    {".f", "#Fortran", 0, 0, 0},
    {".for", "#Fortran", 0, 0, 0},
    {".f90", "#Fortran", 0, 0, 0},
    {".p", "#Pascal", 0, 0, 0},
    {".java", "#Java", 0, 0, 0},
    {".c", "@c", 0, 1, 1},
    {".h", "@c-header", 0, 0, 0},
    {".i", "@cpp-output", 0, 1, 0},
    {".s", "@assembler", 0, 1, 0}
};
```
Default Specs in the Plugin Data Structure in `gcc.c`

All entries of Objective C/C++ and some entries of Fortran removed.

```c
static const struct compiler default_compilers[] = {
    {".cc", "#C++", 0, 0, 0},
    {".cpp", "#C++", 0, 0, 0},
    {".c++", "#C++", 0, 0, 0},
    {".cpp", "#C++", 0, 0, 0},
    {".c++", "#C++", 0, 0, 0},
    {".C", "#C++", 0, 0, 0},
    {".CPP", "#C++", 0, 0, 0},
    {".ii", "#C++", 0, 0, 0},
    {".ads", "#Ada", 0, 0, 0},
    {".f", "#Fortran", 0, 0, 0},
    {".for", "#Fortran", 0, 0, 0},
    {".f90", "#Fortran", 0, 0, 0},
    {".p", "#Pascal", 0, 0, 0},
    {".java", "#Java", 0, 0, 0},
    {".c", "@c", 0, 1, 1},
    {".h", "@c-header", 0, 0, 0},
    {".i", "@cpp-output", 0, 1, 0},
    {".s", "@assembler", 0, 1, 0}
};
```

- @: Aliased entry
All entries of Objective C/C++ and some entries of Fortran removed.

```c
static const struct compiler default_compilers[] = {
    {".cc", "#C++", 0, 0, 0},
    {".cxx", "#C++", 0, 0, 0},
    {".cpp", "#C++", 0, 0, 0},
    {".cp", "#C++", 0, 0, 0},
    {".c++", "#C++", 0, 0, 0},
    {".C", "#C++", 0, 0, 0},
    {".CPP", "#C++", 0, 0, 0},
    {".ii", "#C++", 0, 0, 0},
    {".ads", "#Ada", 0, 0, 0},
    {".adb", "#Ada", 0, 0, 0},
    {".f", "#Fortran", 0, 0, 0},
    {".for", "#Fortran", 0, 0, 0},
    {".f90", "#Fortran", 0, 0, 0},
    {".p", "#Pascal", 0, 0, 0},
    {".java", "#Java", 0, 0, 0},
    {".c", "@c", 0, 1, 1},
    {".h", "@c-header", 0, 0, 0},
    {".i", "@cpp-output", 0, 1, 0},
    {".s", "@assembler", 0, 1, 0},
    {".cxx", "#C++", 0, 0, 0},
    {".cp", "#C++", 0, 0, 0},
    {".C", "#C++", 0, 0, 0},
    {".CPP", "#C++", 0, 0, 0},
    {".ii", "#C++", 0, 0, 0},
    {".adb", "#Ada", 0, 0, 0},
    {".F", "#Pascal", 0, 0, 0},
    {".FOR", "#Pascal", 0, 0, 0},
    {".F90", "#Fortran", 0, 0, 0},
    {".pas", "#Pascal", 0, 0, 0},
    {".class", "#Java", 0, 0, 0},
};
```

- `@`: Aliased entry
- `#`: Default specs not available
Complete Entry for C in gcc.c

{"@c",
 /* cc1 has an integrated ISO C preprocessor. We should invoke the external preprocessor if -save-temps is given. */
 "%{E|M|MM:%(trad_capable_cpp) %(cpp_options) %(cpp_debug_options)}\n %{!E:%{!M:%{!MM:

 %{traditional|ftraditional:}

 %eGNU C no longer supports -traditional without -E}\n
 %{!combine:\n
 %{save-temps|traditional-cpp|no-integrated-cpp:%(trad_capable_cpp) \n %(cpp_options) -o %{save-temps:%b.i} %{!save-temps:%g.i} \n cc1 -fpreprocessed %{save-temps:%b.i} %{!save-temps:%g.i} \n %(cc1_options)}\n
 %{!save-temps:%{!traditional-cpp:%{!no-integrated-cpp:\n cc1 %(cpp_unique_options) %(cc1_options))}}\n
 %{!fsyntax-only:%(invoke_as)}} \n
 %{combine:\n
 %{save-temps|traditional-cpp|no-integrated-cpp:%(trad_capable_cpp) \n %(cpp_options) -o %{save-temps:%b.i} %{!save-temps:%g.i}}\n
 %{!save-temps:%{!traditional-cpp:%{!no-integrated-cpp:\n cc1 %(cpp_unique_options) %(cc1_options))}\n
 %{!fsyntax-only:%(invoke_as)}}"}, 0, 1, 1},
Populated Plugin Data Structure for C++:
gcc/cp/lang-specs.h

{".cc", "@c++", 0, 0, 0},
{".cp", "@c++", 0, 0, 0},
{".cxx", "@c++", 0, 0, 0},
{".cpp", "@c++", 0, 0, 0},
{".c++", "@c++", 0, 0, 0},
{".C", "@c++", 0, 0, 0},
{".CPP", "@c++", 0, 0, 0},
{".H", "@c++-header", 0, 0, 0},
{".hpp", "@c++-header", 0, 0, 0},
{".hp", "@c++-header", 0, 0, 0},
{".hxx", "@c++-header", 0, 0, 0},
{".h++", "@c++-header", 0, 0, 0},
{".HPP", "@c++-header", 0, 0, 0},
{".tcc", "@c++-header", 0, 0, 0},
{".hh", "@c++-header", 0, 0, 0},
Populated Plugin Data Structure for C++:
gcc/cp/lang-specs.h

{"@c++-header",
 @{E|M|MM:cc1plus -E %(cpp_options) %2 %(cpp_debug_options)}\n %{!E:%{!M:{!MM:\n %{save-temps|no-integrated-cpp:cc1plus -E\n
%(cpp_options) %2 -o %{save-temps:%b.ii} %{!save-temps:%g.ii} \n} cc1plus %{save-temps|no-integrated-cpp:-fpreprocessed %{save-temps:%
 %{!save-temps:{!no-integrated-cpp:%(cpp_unique_options)}}\n
%(cc1_options) %2\n
%{!fsyntax-only:%{!fdump-ada-spec*:-o %g.s %{!o*:---output-pch=%i.gch}\n %W{o*:---output-pch=%*} %W} }"},
CPLUSPLUS_CPP_SPEC, 0, 0},

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Populated Plugin Data Structure for C++:
gcc/cp/lang-specs.h

{"@c++",
  "%{E|M|MM:cc1plus -E %(cpp_options) %2 %(cpp_debug_options)}\n  %{!E:%{!M:{!MM:\n    %{save-temps|no-integrated-cpp:cc1plus -E}\n    %(cpp_options) %2 -o %{save-temps:%b.ii} %{!save-temps:%g.ii} \n  cc1plus %{save-temps|no-integrated-cpp:-fpreprocessed %{save-temps:%
    %{!save-temps:%{!no-integrated-cpp:%(cpp_unique_options)}}}\n    %(cc1_options) %2\n    %{!fsyntax-only:%(invoke_as)}}}}",
  CPLPLUSPLUS_CPP_SPEC, 0, 0},
{".ii", "@c++-cpp-output", 0, 0, 0},

{"@c++-cpp-output",
  "%{!M:{!MM:{!E:\n    cc1plus -fpreprocessed %i %(cc1_options) %2\n    %{!fsyntax-only:%(invoke_as)}}}}", 0, 0, 0},
Populated Plugin Data Structure for LTO:
 gcc/lto/lang-specs.h

/* LTO contributions to the "compilers" array in gcc.c. */

{"@lto", "lto1 %(cc1_options) %i %{!fsyntax-only:%(invoke_as)}", /*cpp_spec=*/NULL, /*combinable=*/1, /*needs_preprocessing=*/0},
What about the Files to be Processed by the Linker?

- Linking is the last step
- Every file is passed on to linker unless it is suppressed
- If a translator is not found, input file is assumed to be a file for linker
Plugin Structure in \textit{cc1}

toplev \rightarrow \text{frontend} \rightarrow \text{pass manager}
Plugin Structure in \texttt{cc1}

- toplev
- main
- frontend
- pass manager
- pass 1
- pass 2
- pass expand
- pass n

Double arrow represents control flow whereas single arrow represents pointer or index.
Plugin Structure in $\text{cc1}$

- **toplevel main**
- **frontend**
- **pass manager**
- **pass 1**
- **pass 2**
- **pass expand**
- **pass n**

- Code for pass 1
- Code for pass 2
- Expander code
- Optab table
- Recognizer code
Plugins: Static Plugins in GCC

Plugin Structure in $cc1$

toplev
    main

front end

pass manager

langhook
    ...  

code for language 1

code for language 2

code for language n

pass 1

code for pass 1

pass 2

code for pass 2

pass expand

optab_table

pass n

recognizer code

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Plugin Structure in cc1

1. toplev
   - main

2. frontend

3. pass manager
   - pass 1
   - code for pass 1
   - pass 2
   - code for pass 2
   - pass expand
   - code
   - optab_table
   - pass n
   - code

4. langhook
   - code for language 1
   - code for language 2
   - code for language n
Plugin Structure in \texttt{cc1}

- toplevel
- \texttt{main}
- \texttt{frontend}
- \texttt{passmanager}
- \texttt{langhook}
- \texttt{code for language 1, language 2, \ldots, language n}
- \texttt{code for pass 1, pass 2, \ldots, pass n}
- \texttt{expander code: optab_table}
- \texttt{recognizer code}
Plugin Structure in \texttt{cc1}

- **toplevel**: main
- **frontend**
- **pass manager**

1. **pass 1**
   - code for pass 1

2. **pass 2**
   - code for pass 2

- **expander**
  - code
  - \texttt{optab_table}

- **insn_data**
  - generated code
  - for machine 1

- **recognizer**
  - code

- **langhook**
  - code for language 1
  - code for language 2
  - code for language n

- **code for languages**
  - MD 1
  - MD 2
  - MD n
Plugin Structure in `cc1`

```
toplevel
  main
    frontend
        pass manager
            code for pass 1
            code for pass 2
                ... pass n
                    insn_data
                        generated code
                            optab_table
                                code for machine 2
                                    recognizer
                                        code
```
Plugin Structure in cc1

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Important fields of struct `lang_hooks` instantiated for C

```c
#define LANG_HOOKS_FINISH c_common_finish
#define LANG_HOOKS_EXPAND_EXPR c_expand_expr
#define LANG_HOOKS_PARSE_FILE c_common_parse_file
#define LANG_HOOKS_WRITE_GLOBALS c_write_global_declarations
```
Plugins for Intraprocedural Passes

struct opt_pass
{
    enum opt_pass_type type;
    const char *name;
    bool (*gate) (void);
    unsigned int (*execute) (void);
    struct opt_pass *sub;
    struct opt_pass *next;
    int static_pass_number;
    timevar_id_t tv_id;
    unsigned int properties_required;
    unsigned int properties_provided;
    unsigned int properties_destroyed;
    unsigned int todo_flags_start;
    unsigned int todo_flags_finish;
};

struct gimple_opt_pass
{
    struct opt_pass pass;
};

struct rtl_opt_pass
{
    struct opt_pass pass;
};
Pass variable: all_simple_ipa_passes

struct simple_ipa_opt_pass
{
    struct opt_pass pass;
};
Plugins for Interprocedural Passes across a Translation Unit

Pass variable: all_regular_ipa_passes

struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                           struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                    struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  void (*stmt_fixup) (struct cgraph_node *, gimple *);
  unsigned int function_transform_todo_flags_start;
  unsigned int (*function_transform) (struct cgraph_node *);
  unsigned int (*variable_transform) (struct varpool_node *);
};
## Predefined Pass Lists

<table>
<thead>
<tr>
<th>Pass Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>all_lowering_passes</td>
<td>Lowering</td>
</tr>
<tr>
<td>all_small_ipa_passes</td>
<td>Early optimization passes. Invokes intraprocedural passes over the call graph.</td>
</tr>
<tr>
<td>all_regular_ipa_passes</td>
<td></td>
</tr>
<tr>
<td>all_lto_gen_passes</td>
<td></td>
</tr>
<tr>
<td>all_passes</td>
<td>Intraprocedural passes on GIMPLE and RTL</td>
</tr>
</tbody>
</table>
Registering a Pass as a Static Plugin

1. Write the driver function in your file

2. Declare your pass in file `tree-pass.h`:
   ```c
   extern struct gimple_opt_pass your_pass_name;
   ```

3. Add your pass to the appropriate pass list in `init_optimization_passes()` using the macro `NEXT_PASS`

4. Add your file details to `$SOURCE/gcc/Makefile.in`

5. Configure and build gcc
   (For simplicity, you can make `cc1` only)

6. Debug `cc1` using `ddd/gdb` if need arises
   (For debugging `cc1` from within `gcc`, see:
Part 3

Dynamic Plugins in GCC
Dynamic Plugins

- Supported on platforms that support `-ldl -rdynamic`
- Loaded using `dlopen` and invoked at pre-determined locations in the compilation process
- Command line option
  ```
  -fplugin=/path/to/name.so
  ```
  Arguments required can be supplied as name-value pairs
The Mechanism of Dynamic Plugin

- pass manager
- code for pass
- code for pass
- expander code
- optab_table
- recognizer code for
- code for dynamic plugin
The Mechanism of Dynamic Plugin

Runtime initialization of the appropriate linked list of passes

Made possible by dynamic linking
Specifying an Example Pass

```c
struct simple_ipa_opt_pass pass_plugin = {
{
    SIMPLE_IPA_PASS,  /* name */
    "dynamic_plug",  /* gate */
    execute_pass_plugin,  /* execute */
    NULL,  /* sub */
    NULL,  /* next */
    0,  /* static pass number */
    TV_INTEGRATION,  /* tv_id */
    0,  /* properties required */
    0,  /* properties provided */
    0,  /* properties destroyed */
    0,  /* todo_flags start */
    0};  /* todo_flags end */
};
```
struct register_pass_info pass_info = {
    &(pass_plugin.pass), /* Address of new pass, here, the struct opt_pass field of simple_ipa_opt_pass defined above */
    "pta", /* Name of the reference pass (string in the structure specification) for hooking up the new pass. */
    0, /* Insert the pass at the specified instance number of the reference pass. Do it for every instance if it is 0. */
    PASS_POS_INSERT_AFTER /* how to insert the new pass: before, after, or replace. Here we are inserting our pass the pass named pta */
};
Registering Callback for Our Pass for a Dynamic Plugins

```c
int plugin_init(struct plugin_name_args *plugin_info,
    struct plugin_gcc_version *version)
{
    /* Plugins are activated using this callback */

    register_callback (
        plugin_info->base_name, /* char *name: Plugin name, could be any name. 
                                  plugin_info->base_name gives this filename */
        PLUGIN_PASS_MANAGER_SETUP, /* int event: The event code. Here, setting up a new pass */
        NULL, /* The function that handles the event */
        &pass_info); /* plugin specific data */

    return 0;
}
```
Makefile for Creating and Using a Dynamic Plugin

```makefile
CC = $(INSTALL_D)/bin/gcc
PLUGIN_SOURCES = new-pass.c
PLUGIN_OBJECTS = $(patsubst %.c,%.o,$(PLUGIN_SOURCES ))
GCCPLUGINS_DIR = $(shell $(CC) -print-file-name=plugin)
CFLAGS+= -fPIC -O2
INCLUDE = -Iplugin/include

%.o : %.c
$(CC) $(CFLAGS) $(INCLUDE) -c $<

new-pass.so: $(PLUGIN_OBJECTS)
    $(CC) $(CFLAGS) $(INCLUDE) -shared $^ -o $@

test_plugin: test.c
    $(CC) -fplugin=./new-pass.so $^ -o $@ -fdump-tree-all
```
Part 4

Flow of Control in the Generated Compiler
Walking the Maze of a Large Code Base

- If you use conventional editors such as vi or emacs
  - Use cscope
    - cd $SOURCE
    - cscope -R
  - Use ctags
    - cd $SOURCE
    - ctags -R
    - Make sure you use exeburant-ctags
- Or use IDE such as eclipse
main  /* In file gcc.c */
  validate_all_switches
lookup_compiler
do_spec
  do_spec_2
    do_spec_1  /* Get the name of the compiler */
execute
  pex_init
  pex_run
    pex_run_in_environment
      obj->funcs->exec_child
```c
main  /* In file gcc.c */
    validate_all_switches
lookup_compiler
do_spec
do_spec_2
do_spec_1  /*
execute
    pex_init
    pex_run
    pex_run_in
    obj->func
```

**Observations**

- All compilers are invoked by this driver
- Assembler is also invoked by this driver
- Linker is invoked in the end by default
main

toplev_main    /* In file toplev.c */
decode_options
do_compile
  compile_file
    lang_hooks.parse_file => c_common_parse_file
    lang_hooks.decls.final_write Globals =>
      c_write_global_declarations
    targetm.asm_out.file_end
  finalize
cc1 Top Level Control Flow

main

toplev_main /* In file toplev.c */

decode_options
do_compile

compile_file

lang_hooks.parse_file => c_common_parse_file

lang_hooks.decls.final_write_globals => c_write_global_declarations

targetm.asm_out.file_end

finalize

Observations

• The entire compilation is driven by functions specified in language hooks

• Not a good design!
cc1  Control Flow: Parsing for C

lang_hooks.parse_file => c_common_parse_file
  c_parse_file
    c_parser_translation_unit
      c_parser_external_declaration
        c_parser_declaration_or_fndef
          c_parser_declsdefs /* parse declarations */
          c_parser_compound_statement
        finish_function  /* finish parsing */
          c_genericize
        cgraph_finalize_function
          /* finalize AST of a function */
cc1  **Control Flow: Parsing for C**

lang_hooks.parse_file => c_common_parse_file
  c_parse_file
    c_parser_translation_unit
      c_parser_external_declaration
          c_parser_compound_statement
              finish_function
                c_genericize
                    cgraph_finalize_function
                        /* finalize AST of a function */
    c_parser_compound_statement
    c_parser_translation_unit
    c_parser_translation_unit
    c_parser_translation_unit
    c_parser_translation_unit

**Observations**

- GCC has moved to a recursive descent parser from version 4.1.0
- Earlier parser was generated using Bison specification
Expected Vs. Actual Schematic

- **Expected**
  - `toplev main` → `frontend` → `pass manager`
  - `langhook` → `code for language 1`
  - `GIMPLE passes`
  - `RTL passes`
  - `insn_data generated code for machine 1`
Expected Vs. Actual Schematic

- Toplevel
- Main
- Langhook

- Code for language 1
  - Front end
  - Pass manager
  - GIMPLE passes
  - Pass expand
  - RTL passes

- Code for pass 1
- Code for pass 2
- Expander code
- Optab_table
- Recognizer code

- Insn_data
- Generated code for machine 1

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
lang_hooks.decls.final_write_globals =>
    c_write_global_declarations

cgraph_finalize_compilation_unit

cgraph_analyze_functions /* Create GIMPLE */
cgraph_analyze_function
    gimplify_function_tree
    gimplify_body
    gimplify_stmt
    gimplify_expr

cgraph_lower_function /* Intraprocedural */
tree_lowering_passes
    execute_pass_list (all_lowering_passes)
**cc1** Control Flow: Lowering Passes for C

```c
lang_hooks.decls.final_write_globals =>
    c_write_global_declarations

cgraph_finalize_compilation_unit

cgraph_analyze_functions

gimplify_function_tree

gimplify_body

gimplify_stmt

gimplify_expr

cgraph_lower_function

Observations

- Lowering passes are language independent
- Yet they are being called from a function in language hooks
- Not a good design!
```
## Organization of Passes

<table>
<thead>
<tr>
<th>Order</th>
<th>Task</th>
<th>IR</th>
<th>Level</th>
<th>Pass data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lowering</td>
<td>GIMPLE</td>
<td>Intra</td>
<td>gimple_opt_pass</td>
</tr>
<tr>
<td>2</td>
<td>Optimizations</td>
<td>GIMPLE</td>
<td>Inter</td>
<td>ipa_opt_pass</td>
</tr>
<tr>
<td>3</td>
<td>Optimizations</td>
<td>GIMPLE</td>
<td>Intra</td>
<td>gimple_opt_pass</td>
</tr>
<tr>
<td>4</td>
<td>RTL Generation</td>
<td>GIMPLE</td>
<td>Intra</td>
<td>rtl_opt_pass</td>
</tr>
<tr>
<td>5</td>
<td>Optimization</td>
<td>RTL</td>
<td>Intra</td>
<td>rtl_opt_pass</td>
</tr>
</tbody>
</table>
Control Flow: Optimization and Code Generation Passes

```c
/* Create GIMPLE */
cgraph_analyze_function

/* previous slide */
cgraph_optimize

ipa_passes

execute_ipa_pass_list(all_small_ipa_passes)  /*!<in_lto_p*/
execute_ipa_summary_passes(all_regular_ipa_passes)
execute_ipa_summary_passes(all_lto_gen_passes)
ipa_write_summaries

cgraph_expand_all_functions

cgraph_expand_function

/* Intraprocedural passes on GIMPLE, */

/* expansion pass, and passes on RTL. */

execute_pass_list (all_passes)
```

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
**Control Flow: Optimization and Code Generation Passes**

Cgraph analyze function

.../* Create GIMPLE */
/* previous slide */

Cgraph optimize

Ipapasses

execute_ipa_pass_list(all_small_ipa_passes)

execute_ipa_summary_passes(all_regular_ipa_passes)

execute_ipa_summary_passes(all_lto_gen_passes)

Ipawritesummary

Cgraph expand all

Cgraph expand function

/* Intraprocedural passes on GIMPLE, */
/* expansion pass, and passes on RTL. */

Tree rest of compilation

execute_pass_list(all_passes)

**Observations**

- Optimization and code generation passes are language independent
- Yet they are being called from a function in language hooks
- Not a good design!
## Execution Order in Intraprocedural Passes

<table>
<thead>
<tr>
<th>Pass</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 2</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 3</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 4</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 5</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>
## Execution Order in Intraprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC
# Execution Order in Intraprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Pass 4</th>
<th>Pass 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function 1</td>
<td>Function 2</td>
<td>Function 3</td>
<td>Function 4</td>
<td>Function 5</td>
</tr>
<tr>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
# Execution Order in Intraprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC  
GCC Resource Center, IIT Bombay
## Execution Order in Intraprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution Order in Intraprocedural Passes

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Execution Order in Intraprocedural Passes

Function 1 | Function 2 | Function 3 | Function 4 | Function 5
---|---|---|---|---
Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Intraprocedural Passes

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Execution Order in Intraprocedural Passes

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Execution Order in Intraprocedural Passes

Pass 1  Pass 2  Pass 3  Pass 4  Pass 5

Function 1  Function 2  Function 3  Function 4  Function 5
Execution Order in Intraprocedural Passes

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Intraprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1
Pass 2
Pass 3
Pass 4
Pass 5
Execution Order in Intraprocedural Passes

Pass 1  Pass 2  Pass 3  Pass 4  Pass 5
Function 1  Function 2  Function 3  Function 4  Function 5
Execution Order in Intraprocedural Passes

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
## Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th></th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 2</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 3</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 4</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td>Pass 5</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
</tr>
</tbody>
</table>
# Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Pass 4</th>
<th>Pass 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function 1</td>
<td>Function 2</td>
<td>Function 3</td>
<td>Function 4</td>
<td>Function 5</td>
</tr>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

- In Pass 1, Function 1 is executed before Function 2.
- In subsequent passes, the order remains constant with no changes in execution order.
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>Pass 2</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>Pass 3</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>Pass 4</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
<tr>
<td>Pass 5</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
<td>∅</td>
</tr>
</tbody>
</table>
### Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Execution Order in Interprocedural Passes

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC
GCC Resource Center, IIT Bombay
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Pass 2</th>
<th>Pass 3</th>
<th>Pass 4</th>
<th>Pass 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function 1</td>
<td>Function 2</td>
<td>Function 3</td>
<td>Function 4</td>
<td>Function 5</td>
</tr>
</tbody>
</table>
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
### Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 1</td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Pass 2</td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Pass 3</td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Pass 4</td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Pass 5</td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
<td><a href="#">Diagram</a></td>
</tr>
</tbody>
</table>
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pass 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pass 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Execution Order in Interprocedural Passes

<table>
<thead>
<tr>
<th>Pass 1</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pass 2</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pass 3</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pass 4</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pass 5</th>
<th>Function 1</th>
<th>Function 2</th>
<th>Function 3</th>
<th>Function 4</th>
<th>Function 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution Order in Interprocedural Passes

Function 1  Function 2  Function 3  Function 4  Function 5

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
Execution Order in Interprocedural Passes

Pass 1

Pass 2

Pass 3

Pass 4

Pass 5
cc1 Control Flow: GIMPLE to RTL Expansion (pass_expand)

gimple_expand_cfg
  expand_gimple_basic_block(bb)
  expand_gimple_cond(stmt)
  expand_gimple_stmt(stmt)
    expand_gimple_stmt_1(stmt)
    expand_expr_real_2
      expand_expr /* Operands */
        expand_expr_real
      optab_for_tree_code
    expand_binop /* Now we have rtx for operands */
      expand_binop_directly
        /* The plugin for a machine */
        code=optab_handler(binoptab,mode);
        GEN_FCN
        emit_insn
Part 5

Link Time Optimization
Motivation for Link Time Optimization

- Default `cgraph` creation is restricted to a translation unit (i.e., a single file) ⇒ Interprocedural analysis and optimization is restricted to a single file
- All files (or their equivalents) are available only at link time (assuming static linking)
- LTO enables interprocedural optimizations across different files
Link Time Optimization

- LTO framework supported in GCC-4.6.0
- Use `-f1to` option during compilation
- Generates conventional `.o` files with GIMPLE level information inserted
  Complete translation is performed in this phase
- During linking all object modules are put together and `1to1` is invoked
- `1to1` re-executes optimization passes from the function `cgraph_optimize`

*Basic Idea: Provide a larger call graph to regular ipa passes*
Understanding LTO Framework

```c
main ()
{
    printf ("hello, world\n");
}
```
Assembly Output without LTO Information (1)

```
.file "t0.c"
.section .rodata

.LC0:
.string "hello, world"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
pushl %ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
movl %esp, %ebp
.cfi_def_cfa_register 5
andl $-16, %esp
subl $16, %esp
movl $.LC0, (%esp)
call puts
leave
.cfi_restore 5
.cfi_def_cfa 4, 4
ret
.cfi_endproc
.LFE0:
.size main, .-main
.ident "GCC: (GNU) 4.6.0"
.section .note.GNU-stack,"",@pro
```
Assembly Output with LTO Information (2)

```assembly
.ascii "\007"
.text
.section .gnu.lto_.refs.6a5c5521,"",@progbits
.string "x\234cb‘‘\006b&\006\030"
.string ""
.string ""
.string "t"
.ascii "\b"
.text
.section .gnu.lto_.statics.6a5c5521,"",@progbits
.string "x\234cb‘‘b\300\016@\342\214\020&"
.string ""
.string "\330"
.ascii "\b"
.text
.section .gnu.lto_.decls.6a5c5521,"",@progbits
.string "x\234\225R=0\002A\020}\273w\352\236\247(Q/!\026\\!F–\214\n.ascii "\021A\360\003\254\355\314jG\207\263w\007\334E\2058\311\333\n.ascii "\331\371|s\307\341I\206\320&\251s‘\226t\272\260\210\236(\n.ascii "\260\213\237\242\336\207b{\204}B\222p@\320\277F8\/3
```

Essential Abstractions in GCC  
GCC Resource Center, IIT Bombay
Assembly Output with LTO Information (3)

```assembly
.asci
"/
",254G,204|323j,307\035\207[w,230qN,204|032gB2,33
.asci "\025|\034\365U,241|f,341|033\314,255a,225\376,237#Y,t\.
.asci \\
"|}215\273\276\245{\342|255\374n,f|035b,332\213\236/#\221
.asci \\
"321\253.Y\021q/ \320|310,0166|322\303\305,275,357L,273\3
.asci \\
"017|f,005\227D,267\3400,333\365Z,325_8h,217j,367f,034j,3
.asci "!r\237y[\f,344,231x,302\304,335\342,222,301{\343,317@,204,371|
.asci "\\\211u}p,324\351,252\201\307\213\262\027\3757S,311j,0,257
.asci "\277,275,0207|\376\nLu,246\221\254\n+\3
.asci \\
"007,367|221\244h,003\223\216,350\354\254,016\343
.asci "\033M,210\356,242|272,211|375,352|005,214,320\2320\3
.asci "zx\236t0f,334,237\273,201|350,255\356\}334,017|2,376F,344,20
.asci "v\222,366,006|206,316V,226S,320S,351\232,221\354q6{\23
.asci "|\003,262q,030,362"
.text
.section .gnu.lto_.symtab.6a5c5521,"",@progbits
.string "main"
.string ""
.string ""
.string ""
.string ""
.string ""
.text
section .gnu.lto_.symtab.6a5c5521,"",@progbits
.string "main"
.string ""
.string ""
.string ""
.string ""
.string ""
```
Assembly Output with LTO Information (4)

```assembly
.string ""
.string ""
.string ""
.string ""
.string ""
.string ""
.string "K"
.string ""
.string ""
.text
.section .gnu.lto_.opts,"",@progbits
.string "x\234cb‘\340\002bs\006b‘\002\021r\f\f\273\230\031\030\0"
.ascii "\002\370\tL"
.text
.section .rodata
.LC0:
.string "hello, world"
```
Assembly Output with LTO Information (5)

.text
.globl main
.type main, @function

main:
.LFB0:
    .cfi_startproc
    pushl %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl %esp, %ebp
    .cfi_def_cfa_register 5
    andl $-16, %esp
    subl $16, %esp
    movl $.LC0, (%esp)
call puts
Assembly Output with LTO Information (6)

leave
.cfi_restore 5
.cfi_def_cfa 4, 4
ret
.cfi_endproc

.LFE0:

.size main, .-main
.comm __gnu_lto_v1,1,1
.ident "GCC: (GNU) 4.6.0"
.section .note.GNU-stack,"",@progbits
Whole program optimization needs to see the entire program

- Does it need the entire program *together* in the memory?

- Load only the call graph without function bodies
  - Independent computation of summary information of functions
  - “Adjusting” summary information through whole program analysis over the call graph
  - Perform transformation independently on functions

**Multi process LTO**

- Process the entire program together

**Single process LTO**
Why Avoid Loading Function Bodies?

- Practical programs could be rather large and compilation could become very inefficient

- Many optimizations decisions can be taken by looking at the call graph alone
  - Procedure Inlining: just looking at the call graph is sufficient
    - Perhaps some summary size information can be used
  - Procedure Cloning: some additional summary information about actual parameters of a call is sufficient
Multi Process LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Local generation of summary information and translation unit information
  - WPA: Whole Program Analysis
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations
Multi Process LTO (aka WHOPR Mode of LTO)

• Three steps
  ▶ LGEN: Local generation of summary information and translation unit information Potentially Parallel
  ▶ WPA: Whole Program Analysis
    – Reads the call graph and not function bodies
    – Summary information for each function
  ▶ LTRANS: Local Transformations
Multi Process LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Local generation of summary information and translation unit information [*Potentially Parallel*]
  - WPA: Whole Program Analysis [*Sequential*]
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations
Multi Process LTO (aka WHOPR Mode of LTO)

• Three steps
  ▶ LGEN: Local generation of summary information and translation unit information **Potentially Parallel**
  ▶ WPA: Whole Program Analysis **Sequential**
    – Reads the call graph and not function bodies
    – Summary information for each function
  ▶ LTRANS: Local Transformations **Potentially Parallel**
Multi Process LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Local generation of summary information and translation unit information **Potentially Parallel**
  - WPA: Whole Program Analysis **Sequential**
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations **Potentially Parallel**

- Why do we call this LTO *Multi Process* LTO?
Multi Process LTO (aka WHOPR Mode of LTO)

- Three steps
  - LGEN: Local generation of summary information and translation unit information Potentially Parallel
  - WPA: Whole Program Analysis Sequential
    - Reads the call graph and not function bodies
    - Summary information for each function
  - LTRANS: Local Transformations Potentially Parallel

- Why do we call this LTO *Multi Process* LTO?
  - gcc executes LGEN
  - Subsequent process of lto1 executes WPA
  - Subsequent independent processes of lto1 execute LTRANS
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary)
  - IPA: Inter-Procedural Analysis
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary) Potentially Parallel
  - IPA: Inter-Procedural Analysis
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary) **Potentially Parallel**
  - IPA: Inter-Procedural Analysis **Sequential**
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary) **Potentially Parallel**
  - IPA: Inter-Procedural Analysis **Sequential**
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations **Sequential**
Three steps

- **LGEN**: Local Generation of translation unit information (no summary) *Potentially Parallel*
- **IPA**: Inter-Procedural Analysis *Sequential*
  - Reads the call graph and function bodies
- **LTRANS**: Local Transformations *Sequential*

Why do we call this LTO *Single Process LTO*?
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary) Potentially Parallel
  - IPA: Inter-Procedural Analysis Sequential
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations Sequential
- Why do we call this LTO Single Process LTO?
  - gcc executes LGEN
  - Subsequent process of lto1 executes both IPA and LTRANS
Single Process LTO

- Three steps
  - LGEN: Local Generation of translation unit information (no summary) \textit{Potentially Parallel}
  - IPA: Inter-Procedural Analysis \textit{Sequential}
    - Reads the call graph and function bodies
  - LTRANS: Local Transformations \textit{Sequential}

- Why do we call this LTO \textit{Single Process} LTO?
  - \texttt{gcc} executes LGEN
  - Subsequent process of lto1 executes both IPA and LTRANS

- When \texttt{-flto-partition=none}, IPA = WPA
LTO Pass Hooks

```c
struct ipa_opt_pass_d {
    struct opt_pass pass;
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                           struct varpool_node_set_def *);
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                       struct varpool_node_set_def *);
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *);
    unsigned int function_transform_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *);
};
```
struct ipa_opt_pass_d
{
    struct opt_pass pass;
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                           struct varpool_node_set_def *);
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                        struct varpool_node_set_def *);
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *);
    unsigned int function_transform_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *);
};
LTO Pass Hooks

```c
struct ipa_opt_pass_d {
    struct opt_pass pass;
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                          struct varpool_node_set_def *);
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                        struct varpool_node_set_def *);
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *);
    unsigned int function_transform_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *);
};
```

LGEN for Single Process LTO
struct ipa_opt_pass_d
{
    struct opt_pass pass; // member void (*execute) (void);
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                           struct varpool_node_set_def *);
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                       struct varpool_node_set_def *);
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *);
    unsigned int function_transform_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *);
};

WPA for Multi Process LTO
struct ipa_opt_pass_d
{
    struct opt_pass pass; (member  void (*execute) (void));
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                            struct varpool_node_set_def *
                           );
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                            struct varpool_node_set_def *
                                           );
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *
                        );
    unsigned int function_transform_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *
                                );
};
LTO Pass Hooks

```c
struct ipa_opt_pass_d
{
  struct opt_pass pass;
  void (*generate_summary) (void);
  void (*read_summary) (void);
  void (*write_summary) (struct cgraph_node_set_def *,
                         struct varpool_node_set_def *);
  void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                     struct varpool_node_set_def *);
  void (*read_optimization_summary) (void);
  void (*stmt_fixup) (struct cgraph_node *, gimple *);
  unsigned int function_transform_todo_flags_start;
  unsigned int (*function_transform) (struct cgraph_node *);
  void (*variable_transform) (struct varpool_node *);
};
```

LTRANS for Multi Process LTO
struct ipa_opt_pass_d {
    struct opt_pass pass;
    void (*generate_summary) (void);
    void (*read_summary) (void);
    void (*write_summary) (struct cgraph_node_set_def *,
                            struct varpool_node_set_def *);
    void (*write_optimization_summary)(struct cgraph_node_set_def *,
                                       struct varpool_node_set_def *);
    void (*read_optimization_summary) (void);
    void (*stmt_fixup) (struct cgraph_node *, gimple *);
    unsigned int function_transform_todo_todo_flags_start;
    unsigned int (*function_transform) (struct cgraph_node *);
    void (*variable_transform) (struct varpool_node *);
};

LTRANS for Single Process LTO
## LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Call graph without function bodies</td>
<td>Call graph with function bodies</td>
</tr>
<tr>
<td>Single partition of the program</td>
<td>Call graph with function bodies</td>
<td>Call graph without function bodies</td>
</tr>
<tr>
<td>Multiple partitions of the program</td>
<td>Call graph with function bodies</td>
<td>Call graph without function bodies</td>
</tr>
</tbody>
</table>

### Whole Program Analysis

- **Call graph without function bodies**
- **Call graph with function bodies**
# LTO Support in GCC

<table>
<thead>
<tr>
<th>Whole Program Analysis</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In the same process as that of analysis</td>
</tr>
<tr>
<td></td>
<td>In an independent process (possibly multiple processes)</td>
</tr>
<tr>
<td></td>
<td>Single partition of the program</td>
</tr>
<tr>
<td></td>
<td>Single partition of the program</td>
</tr>
<tr>
<td></td>
<td>Multiple partitions of the program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Call graph without function bodies</th>
<th>Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call graph with function bodies</td>
<td></td>
</tr>
</tbody>
</table>

**Essential Abstractions in GCC**

**GCC Resource Center, IIT Bombay**
## LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
<tr>
<td>Whole Program Analysis</td>
<td>Call graph without function bodies</td>
<td>Not supported</td>
</tr>
<tr>
<td>Call graph with function bodies</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
</tbody>
</table>

### Essential Abstractions in GCC

**Whole Program Analysis**

- **Call graph without function bodies**
  - Not supported in GCC-4.6.0
  - Will be supported in future

- **Call graph with function bodies**
### LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Whole Program Analysis</th>
<th>Call graph without function bodies</th>
<th>Call graph with function bodies</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Not supported</td>
<td>Supported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td>Suppported in GCC-4.6.0</td>
<td>Will be suppported in future</td>
</tr>
</tbody>
</table>
# LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Single partition of the program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Single partition of the program</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Multiple partitions of the program</td>
</tr>
<tr>
<td><strong>Whole Program Analysis</strong></td>
<td>Call graph without function bodies</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will be supported in future</td>
</tr>
<tr>
<td>Call graph with function bodies</td>
<td>Supppported in GCC-4.6.0</td>
<td>Not supppported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not supppported</td>
</tr>
</tbody>
</table>
# LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
<tr>
<td>Whole Program Analysis</td>
<td>Call graph without function bodies</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td>Call graph with function bodies</td>
<td>Supported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will be supported in future</td>
</tr>
<tr>
<td>-flto</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
</tr>
<tr>
<td></td>
<td>Multiple partitions of the program</td>
<td></td>
</tr>
<tr>
<td>Whole Program Analysis</td>
<td>Call graph without function bodies</td>
<td>Not supported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Supppported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will be supppported in future</td>
</tr>
<tr>
<td>Call graph with function bodies</td>
<td>Suppported in GCC-4.6.0</td>
<td>Not suppported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not suppported</td>
</tr>
</tbody>
</table>

**-flto**

**WHOPR mode**
## LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single partition of the program</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiple partitions of the program</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Whole Program Analysis

<table>
<thead>
<tr>
<th>Call graph without function bodies</th>
<th>-flto</th>
<th>-flto -flto-partition=none</th>
<th>WHOPR mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Whole Program Analysis</td>
<td>Not supported</td>
<td>Supported in GCC-4.6.0</td>
<td>Will be supported in future</td>
</tr>
<tr>
<td>Call graph with function bodies</td>
<td>Supported in GCC-4.6.0</td>
<td>Not supported</td>
<td>Not supported</td>
</tr>
</tbody>
</table>
# LTO Support in GCC

<table>
<thead>
<tr>
<th>Transformation</th>
<th>In the same process as that of analysis</th>
<th>In an independent process (possibly multiple processes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single partition of the program</td>
<td>Single partition of the program</td>
<td>Multiple partitions of the program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Whole Program Analysis</th>
<th>Call graph without function bodies</th>
<th>Not supported</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Call graph with function bodies</td>
<td>Supported in GCC-4.6.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Will be supported in future</td>
</tr>
<tr>
<td>-flto</td>
<td>-flto -flto-partition=none</td>
<td>WHOPR mode</td>
</tr>
</tbody>
</table>

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
lto1 Control Flow

```c
lto_main
    lto_process_name
    lto_init_reader
    read_cgraph_and_symbols
        if (flag_wpa)
            /* WPA for multi process LTO */
            do_whole_program_analysis
                materialize_cgraph
                execute_ipa_pass_list (all_regular_ipa_passes)
                    lto_wpa_write_files
        else
            /* WPA and LTRANS for single process LTO */
            /* Only LTRANS for multi process LTO */
            materialize_cgraph
                cgraph_optimize
```
**Control Flow: A Recap**

toplev_main  /* In file toplev.c */  
compile_file  
lang_hooks.parse_file=>c_common_parse_file  
lang_hooks.decls.final_write Globals=>c_write_global_declarations  
cgraph_finalize_compilation_unit  
cgraph_analyze_functions  /* Create GIMPLE */  
cgraph_analyze_function  /* Create GIMPLE */  
...  
cgraph_optimize  
ipa_passes  
execute_ipa_pass_list(all_small_ipa_passes) /*!in lto*/  
execute_ipa_summary_passes(all_regular_ipa_passes)  
execute_ipa_summary_passes(all_lto_gen_passes)  
ipa_write_summaries  
cgraph_expand_all_functions  
cgraph_expand_function  
/* Intraprocedural passes on GIMPLE, */  
/* expansion pass, and passes on RTL. */
cc1 and Single Process lto1

toplev_main
...

compile_file
...

cgraph_analyze_function

cgraph_optimize
...

ipa_passes
...

cgraph_expand_all_functions
...

tree_rest_of_compilation
cc1 and Single Process lto1

toplev_main
... compile_file
... cgraph_analyze_function

cgraph_optimize
... ipa_passes
... cgraph_expand_all_functions
... tree_rest_of_compilation

lto_main
... read_cgraph_and_symbols
... materialize_cgraph

lto1
### Our Pictorial Convention

<table>
<thead>
<tr>
<th>Source code</th>
<th>cc1'</th>
<th>lto1'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>common</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cc1 executable</th>
<th>cc1'</th>
<th>lto1'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>common</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lto1 executable</th>
<th>cc1'</th>
<th>lto1'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>common</td>
</tr>
</tbody>
</table>
The GNU Tool Chain: Our First Picture

Source Program

```
gcc
```

Target Program

```
cc1
```
```
cpp
```
```
as
```
```
l1d
```
```
glibc/newlib
```
The GNU Tool Chain: Our First Picture

Source Program

gcc

Target Program

cc1

cpp

as

ld

glibc/newlib

via collect2
The GNU Tool Chain for Single Process LTO Support
The GNU Tool Chain for Single Process LTO Support

- gcc
- cc1′
- lto1′
- common
- cc1
The GNU Tool Chain for Single Process LTO Support

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The GNU Tool Chain for Single Process LTO Support

```
cc1'  lto1'
common

"Fat" .s files

as

"Fat" .o files
```
The GNU Tool Chain for Single Process LTO Support

- gcc
- cc1
- lto1
- common
- "Fat".s files
- "Fat".o files
- as
- collect2
The GNU Tool Chain for Single Process LTO Support
The GNU Tool Chain for Single Process LTO Support

- gcc
- cc1
- lto1
- as
- collect2
- common
- "Fat" .s files
- "Fat" .o files
- Single .s file

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The GNU Tool Chain for Single Process LTO Support

Essential Abstractions in GCC
The GNU Tool Chain for Single Process LTO Support

gcc

cc1

lto1

collect2

as

"Fat" .s files

"Fat" .o files

Single .s file

Single .o file

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The GNU Tool Chain for Single Process LTO Support

```
cc1
common
```

```
cc1'
```

```
lto1'
```

```
"Fat".s files
```

```
"Fat".o files
```

```
cc1
```

```
common
```

```
as
```

```
Single .s file
```

```
lto1
```

```
common
```

```
as
```

```
Single .o file + glibc/newlib
```

```
collect2
```

```
ld
```

```
a.out file
```

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
The GNU Tool Chain for Single Process LTO Support

Common Code (executed twice for each function in the input program for single process LTO. Once during LGEN and then during WPA + LTRANS)

cgraph_optimize
    ipa_passes
        execute_ipa_pass_list(all_small_ipa_passes) /*!in lto*/
        execute_ipa_summary_passes(all_regular_ipa_passes)
        execute_ipa_summary_passes(all_lto_gen_passes)
        ipa_write_summaries
    cgraph_expand_all_functions
        cgraph_expand_function
        /* Intraprocedural passes on GIMPLE, */
        /* expansion pass, and passes on RTL. */

a.out file
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

\[ f1.c \rightarrow ccl' \rightarrow lto1' \rightarrow \text{common} \rightarrow f1.o \]
Multi Process LTO (aka WHOPR LTO)

Option \(-f\text{lto} -c\)

\[ f1.c \rightarrow \text{cc1'} \rightarrow \text{lto1'} \rightarrow f1.o \]

\[ f2.c \rightarrow \text{cc1'} \rightarrow \text{lto1'} \rightarrow f2.o \]
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

f1.c → `ccl'` → `lto1'` → `common` → f1.o

f2.c → `ccl'` → `lto1'` → `common` → f2.o

f3.c → `ccl'` → `lto1'` → `common` → f3.o
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

```
f1.c → ccl' → lto' → f1.o
      common
```

Option `-flto -o out`

```
f2.c → ccl' → lto' → f2.o → f2.o
      common
```

```
f3.c → ccl' → lto' → f3.o
      common
```
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

```
f1.c  →  cc1'  →  lto1'  →  f1.o
       common
```

```
f2.c  →  cc1'  →  lto1'  →  f2.o  →  cc1'  →  lto1'
       common
```

```
f3.c  →  cc1'  →  lto1'  →  f3.o
       common
```

Option `-flto -o out`

```
       out
```
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

f1.c → ccl' → lto' → common → f1.o

f2.c → ccl' → lto' → common → f2.o

f3.c → ccl' → lto' → common → f3.o

Option `-flto -o out`

External View

Internal View

out
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

f1.c → `cc1` → `lto1` → common → f1.o

f2.c → `cc1` → `lto1` → common → f2.o

f3.c → `cc1` → `lto1` → common → f3.o

Option `-flto -o out`

```
/bsd/ccda/comp/leto/standard-impl/ld/extern.c
```

External View

Internal View

large call graph without procedure bodies
(Interproc. analysis: √
Tranformation: ×)

`/tmp/ccdKEyVB.ltrans0.o`
(possibly multiple files)
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

```
f1.c  →  cc1'  →  lto1'  →  f1.o
        common
```

```
f2.c  →  cc1'  →  lto1'  →  f2.o  →  cc1'  →  lto1'  →  common
        common
```

```
f3.c  →  cc1'  →  lto1'  →  f3.o
        common
```

Option `-flto -o out`

```
of1.o  →  lto1'  →  common
```

```
of2.o  →  cc1'  →  lto1'  →  common
```

```
of3.o  →  cc1'  →  lto1'  →  common
```

```
out    ←  cc1'  →  lto1'  →  common
```

External View

Internal View

large call graph without procedure bodies (Interproc. analysis: √ Transformation: ×)

/tmp/ccdKEyVB.ltrans0.o (possibly multiple files)
Multi Process LTO (aka WHOPR LTO)

1. f1.c → cc1 → lto1 → f1.o
   - Option `-flto -c`
   - LTO

2. f2.c → cc1 → lto1 → f2.o
   - LGEN

3. f3.c → cc1 → lto1 → f3.o

Option `-flto -o out`

- Large call graph without procedure bodies
- (Interproc. analysis: √ Transformation: ×)

 /	mp/ccdKEyVB.ltrans0.o (possibly multiple files)

Essential Abstractions in GCC
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

f1.c → c1' → lto1' → f1.o

f2.c → c1' → lto1' → f2.o

f3.c → c1' → lto1' → f3.o

Option `-flto -o out`

large call graph without procedure bodies (Interproc. analysis: √ Transformation: ×)

/tmp/ccdKEyVB.ltrans0.o (possibly multiple files)

LGEN

WPA

out
Multi Process LTO (aka WHOPR LTO)

Option `-flto -c`

f1.c → `cc1’` → `lto1’` → f1.o

f2.c → `cc1’` → `lto1’` → f2.o

f3.c → `cc1’` → `lto1’` → f3.o

Option `-flto -o out`

```
/fmp/ccdKEyVB.ltrans0.o
```

WPA

large call graph without procedure bodies
(Interproc. analysis: √
Transformation: ×)

LGEN

LTRANS

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
**Single Process LTO**

Option `-flto -c`

```plaintext
f1.c → ccl' → lto1' → common → f1.o
```
Single Process LTO

Option `-flto -c`

\[
\begin{align*}
\text{f1.c} & \rightarrow \text{cc1' lto1' common} \rightarrow \text{f1.o} \\
\text{f2.c} & \rightarrow \text{cc1' lto1' common} \rightarrow \text{f2.o}
\end{align*}
\]
Single Process LTO

Option `-flto -c`

```
f1.c →  cc1' lto1' common → f1.o
f2.c →  cc1' lto1' common → f2.o
f3.c →  cc1' lto1' common → f3.o
```
Single Process LTO

Option `-flto -c`

f1.c → \texttt{cc1} → \texttt{lto1} → \texttt{f1.o}

f2.c → \texttt{cc1} → \texttt{lto1} → \texttt{f2.o}

f3.c → \texttt{cc1} → \texttt{lto1} → \texttt{f3.o}

Option

`-flto -o out`

`-flto-partition=none`

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Single Process LTO

Option `-f lto -c`

f1.c → `cc1` → `lto1` → common → f1.o

f2.c → `cc1` → `lto1` → common → f2.o

f3.c → `cc1` → `lto1` → common → f3.o

Option
- `-f lto -o out`
- `-f lto-partition=none`

out
Single Process LTO

Option `-flto -c`

External View

Option

- `-flto -o out`
- `-flto-partition=none`

Internal View

Essential Abstractions in GCC
Single Process LTO

Option `-flto -c`

f1.c → `cc1'` → `lto1'` → `f1.o`

```
Option
-flto -c
```

f2.c → `cc1'` → `lto1'` → `f2.o`

```
-flto -o out
-flto-partition=none
```

f3.c → `cc1'` → `lto1'` → `f3.o`

```
out
```

External View

Internal View

large call graph
with procedure bodies
(Interproc. analysis: √
Transformation: √)

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Single Process LTO

Option `-flto -c`

External View

Option
- `-flto -o out`
- `-flto-partition=none`

Internal View

large call graph with procedure bodies (Interproc. analysis: √ Transformation: √)

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Single Process LTO

Example:
- **f1.c**
  - cc1
  - lto1
  - common
  - flto
  - f1.o

- **f2.c**
  - cc1
  - lto1
  - common
  - flto
  - f2.o

- **f3.c**
  - cc1
  - lto1
  - common
  - flto
  - f3.o

Option `-flto -c`

Option `-flto -o out -flto-partition=none`

Large call graph with procedure bodies (Interproc. analysis: \( \sqrt{\) Transformation: \( \sqrt{\) )

LGEN

Essential Abstractions in GCC 

GCC Resource Center, IIT Bombay
Single Process LTO

Option \(-\text{flto} -c\)

\begin{align*}
\text{f1.c} & \rightarrow \text{ccl'} \rightarrow \text{lto1'} \rightarrow \text{f1.o} \\
\text{f2.c} & \rightarrow \text{ccl'} \rightarrow \text{lto1'} \rightarrow \text{f2.o} \\
\text{f3.c} & \rightarrow \text{ccl'} \rightarrow \text{lto1'} \rightarrow \text{f3.o}
\end{align*}

IPA + LTRANS

Option
\begin{align*}
-\text{flto} -o \text{out} \\
-\text{flto-partition}=\text{none}
\end{align*}

large call graph with procedure bodies (Interproc. analysis: √ Transformation: √)

Option -flto -c

LGEN

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Single Process LTO

Option `-flto -c`

f1.c → cc1′ → lto1′ → common → f1.o

f2.c → cc1′ → lto1′ → common → f2.o

f3.c → cc1′ → lto1′ → common → f3.o

IPA + LTRANS

Option

-`flto -o out`
-`-flto-partition=none`

large call graph with procedure bodies (Interproc. analysis: √ Transformation: √)

This WPA can examine function bodies also

out

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay
Part 6

Conclusions
Conclusions

- Excellent mechanism of plugging in different
  - translators in the main driver
  - front ends, passes, and back ends in the main compiler
- However, the plugins have been used in an adhoc manner
- LTO is a good support for interprocedural analysis and optimization
  It would be useful to support
  - a single process LTO mode that
  - creates a large call graph of the entire program with
  - on-demand loading of procedure bodies for
  - enabling examining procedure bodies for interprocedural analysis