Workshop on Essential Abstractions in GCC

Manipulating GIMPLE and RTL IRs

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

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



1 July 2011

GIMPLE and RTL: Outline
Outline

- An Overview of GIMPLE
- Using GIMPLE API in GCC-4.6.0
- Adding a GIMPLE Pass to GCC
- An Internal View of RTL
- Manipulating RTL IR

Essential Abstractions in GCC	GCC Resource Center, II	T Bombay
1 July 2011	GIMPLE and RTL: An Overview of GIMPLE	2/45
GIMPLE: A Recap		

Part 1

An Overview of GIMPLE

- Language independent three address code representation
 - Computation represented as a sequence of basic operations
 - Temporaries introduced to hold intermediate values
- Control construct explicated into conditional and unconditional jumps



GIMPLE and RTL: An Overview of GIMPLE

Motivation Behind GIMPLE

1 July 2011

3/45

4/45

Why Not Abstract Syntax Trees for Optimization?

- Previously, the only common IR was RTL (Register Transfer Language)
- Drawbacks of RTL for performing high-level optimizations
 - Low-level IR, more suitable for machine dependent optimizations (e.g., peephole optimization)
 - High level information is difficult to extract from RTL (e.g. array references, data types etc.)
 - Introduces stack too soon, even if later optimizations do not require it

- ASTs contain detailed function information but are not suitable for optimization because
 - Lack of a common representation across languages
 - No single AST shared by all front-ends
 - So each language would have to have a different implementation of the same optimizations
 - Difficult to maintain and upgrade so many optimization frameworks
 - Structural Complexity
 - Lots of complexity due to the syntactic constructs of each language
 - Hierarchical structure and not linear structure Control flow explication is required



- Earlier versions of GCC would build up trees for a single statement, and then lower them to RTL before moving on to the next statement
- For higher level optimizations, entire function needs to be represented in trees in a language-independent way.
- Result of this effort GENERIC and GIMPLE

What?

- Language independent IR for a complete function in the form of trees
- Obtained by removing language specific constructs from ASTs
- All tree codes defined in \$(SOURCE)/gcc/tree.def

Why?

- Each language frontend can have its own AST
- Once parsing is complete they must emit GENERIC

GIMPLE and RTL: An Overview of GIMPLE

What is **GIMPLE** ?

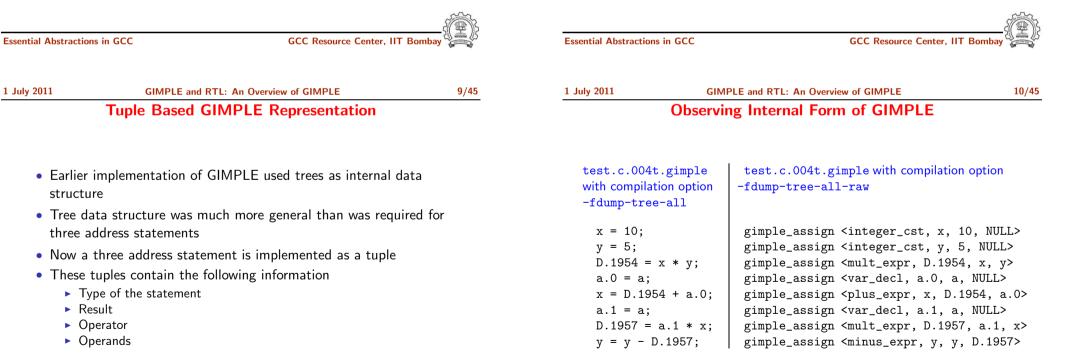
8/45

GIMPLE Goals

- GIMPLE is influenced by SIMPLE IR of McCat compiler
- But GIMPLE is not same as SIMPLE (GIMPLE supports GOTO)
- It is a simplified subset of GENERIC
 - 3 address representation
 - Control flow lowering
 - Cleanups and simplification, restricted grammar
- Benefit : Optimizations become easier

The Goals of GIMPLE are

- Lower control flow Sequenced statements + conditional and unconditional jumps
- Simplify expressions
 Typically one operator and at most two operands
- Simplify scope Move local scope to block begin, including temporaries



The result and operands are still represented using trees



GIMPLE and RTL: An Overview of GIMPLE

11/45

Observing Internal Form of GIMPLE

test.c.004t.gimple with compilation option -fdump-tree-all	<pre>test.c.004t.gimple with compilation option -fdump-tree-all-raw</pre>
<pre>if (a < c) goto <d.1953>; else goto <d.1954>; <d.1953>: a = b + c; goto <d.1955>; <d.1954>: a = b - c; <d.1955>:</d.1955></d.1954></d.1955></d.1953></d.1954></d.1953></pre>	<pre>gimple_cond <lt_expr, a,c,<d.1953="">, <d.1954>> gimple_label <<d.1953>> gimple_assign <plus_expr, a,="" b,="" c=""> gimple_goto <<d.1955>> gimple_label <<d.1954>> gimple_assign <minus_expr, a,="" b,="" c=""> gimple_label <<d.1955>></d.1955></minus_expr,></d.1954></d.1955></plus_expr,></d.1953></d.1954></lt_expr,></pre>



Using GIMPLE API in GCC-4.6.0

	Iterating Over GIMPLE Statements	
1 July 2011	GIMPLE and RTL: Using GIMPLE API in GCC-4.6.0	12/45
Essential Abstracti	ons in GCC GCC Resource Center, IIT	Bombay Com
		Rambar (

• A basic block contains a doubly linked-list of GIMPLE statements

• The statements are represented as GIMPLE tuples, and the

operands are represented by tree data structure

• Processing of statements can be done through iterators

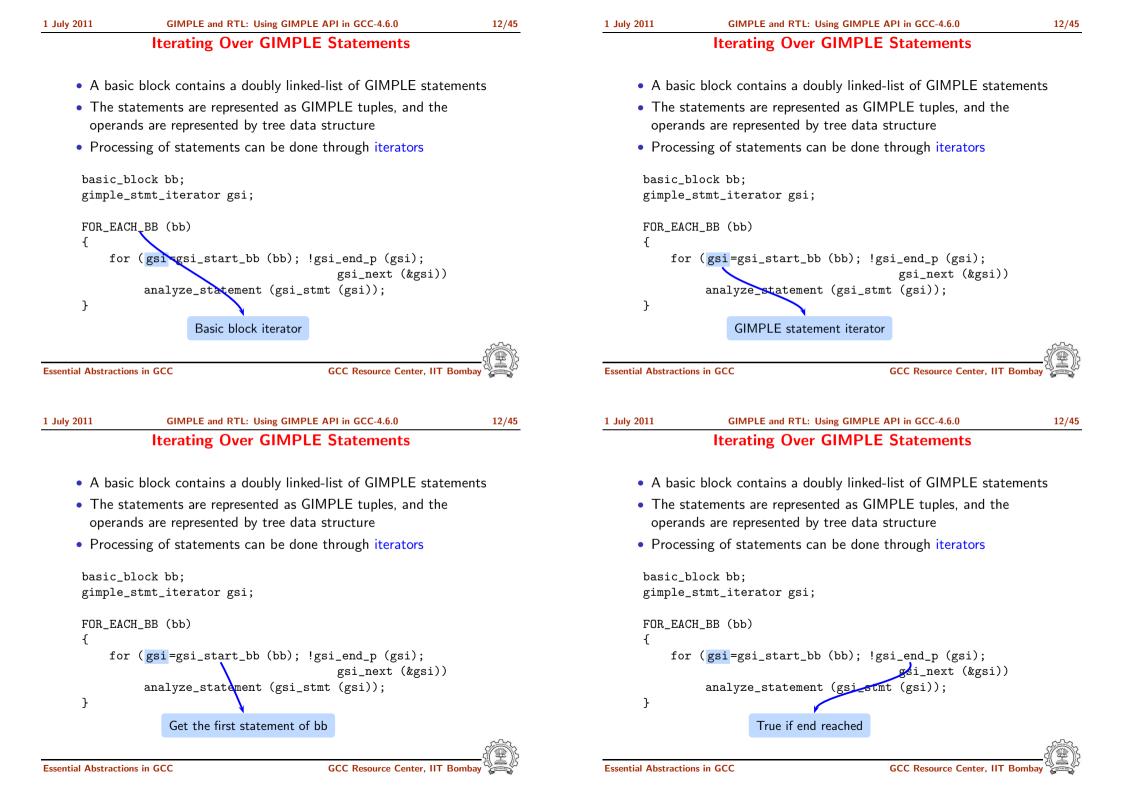
1 July 2011	GIMPLE and RTL: Using GIMPLE API in GCC-4.6.0	12/45
	Iterating Over GIMPLE Statements	

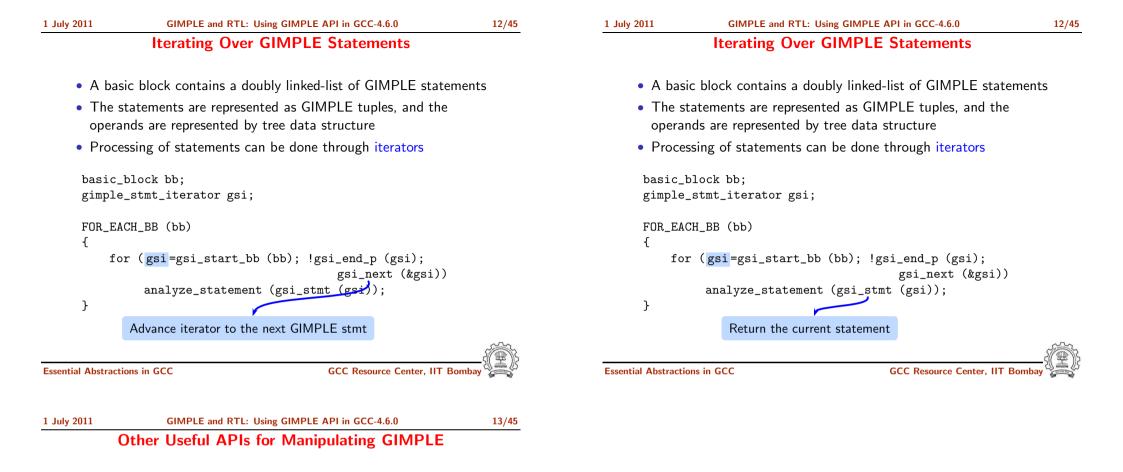
- A basic block contains a doubly linked-list of GIMPLE statements
- The statements are represented as GIMPLE tuples, and the operands are represented by tree data structure
- Processing of statements can be done through iterators

```
basic_block bb;
gimple_stmt_iterator gsi;
```









Extracting parts of GIMPLE statements:

- gimple_assign_lhs: left hand side
- gimple_assign_rhs1: left operand of the right hand side
- gimple_assign_rhs2: right operand of the right hand side
- gimple_assign_rhs_code: operator on the right hand side

A complete list can be found in the file ${\tt gimple.h}$

Part 3

Adding a GIMPLE Pass to GCC

GIMPLE and RTL: Adding a GIMPLE Pass to GCC

Adding a GIMPLE Intraprocedural Pass in GCC-4.6.0 (1)

Add the following gimple_opt_pass struct instance to the file struct gimple_opt_pass pass_intra_gimple_manipulation = {

{	
GIMPLE_PASS,	<pre>/* optimization pass type */</pre>
"gm",	/* name of the pass*/
gate_gimple_manipulation,	/* gate. */
intra_gimple_manipulation,	<pre>/* execute (driver function) */</pre>
NULL,	<pre>/* sub passes to be run */</pre>
NULL,	/* next pass to run */
0,	/* static pass number */
0,	/* timevar_id */
0,	<pre>/* properties required */</pre>
0,	<pre>/* properties provided */</pre>
0,	<pre>/* properties destroyed */</pre>
0,	/* todo_flags start */
0	/* todo_flags end */
}	~~~
};	
ential Abstractions in GCC	GCC Resource Center, IIT Bombay

1 July	2011
--------	------

Esse

GIMPLE and RTL: Adding a GIMPLE Pass to GCC

Adding a GIMPLE Intraprocedural Pass as a Static Plugin

- In \$SOURCE/gcc/Makefile.in, add new-pass.o to the list of language independent object files. Also, make specific changes to compile new-pass.o from new-pass.c
- Configure and build gcc (For simplicity, we will make cc1 only)
- 6. Debug cc1 using ddd/gdb if need arises (For debuging cc1 from within gcc, see: http://gcc.gnu.org/ml/gcc/2004-03/msg01195.html)

- 1 July 2011 GIMPLE and RTL: Adding a GIMPLE Pass to GCC 15/45 Adding a GIMPLE Intraprocedural Pass as a Static Plugin 1. Write the driver function in file new-pass.c 2. Declare your pass in file tree-pass.h: extern struct gimple_opt_pass pass_intra_gimple_manipulation; 3. Add your pass to the intraprocedural pass list in init_optimization_passes() . . . NEXT_PASS (pass_build_cfg); NEXT_PASS (pass_intra_gimple_manipulation); . . . Essential Abstractions in GCC GCC Resource Center, IIT 1 July 2011 GIMPLE and RTL: Adding a GIMPLE Pass to GCC 16/45 **Registering Our Pass as a Dynamic Plugin** struct register_pass_info dynamic_pass_info = { &(pass_intra_gimple_manipulation.pass), /* Address of new pass, here, the struct opt_pass field of simple_ipa_opt_pass defined above */ "cfg" /* Name of the reference pass (string in the pass 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

};



14/45



1 July 2011

{ /* Plugins are activiated 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 */
	&dynamic_pass_info);	/*	plugin specific data */
et	urn 0:		

return 0;

}

Essential Abstractions in GCC

GCC Resource Center, IIT Bombay

19/45

1 July 2011	GIMPLE and RTL: Adding a GIMPLE Pass to GCC	1
An Intra	procedural Analysis for Discovering Pointer	Usage

Calculate the number of pointer statements in GIMPLE (i.e. result or an operand is a pointer variable)



GIMPLE and RTL: Adding a GIMPLE Pass to GCC

Makefile for Creating and Using a Dynamic Plugin

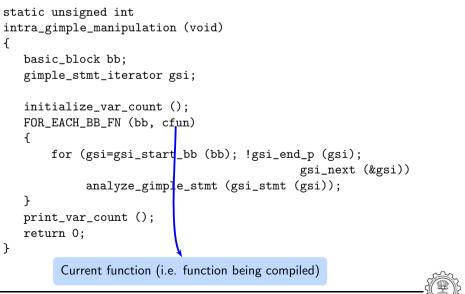
-	s.c bst %.c,%.o,\$(PLUGIN_SOURCES)) \$(CC) -print-file-name=plugin)
%.0 : %.c \$(CC) \$(CFLAGS) \$(INCLUE	E) -c \$<
new-pass.so: \$(PLUGIN_OE \$(CC) \$(CFLAGS)	JECTS) \$(INCLUDE) -shared \$^ -o \$@
test_plugin: test.c \$(CC) -fplugin=.	/new-pass.so \$^ -o \$@ -fdump-tree-all
Essential Abstractions in GCC	GCC Resource Center, IIT Bombay

1 July 2011 GIMPLE and R	TL: Adding a GIMPLE Pass to GCC 20/4
Discove	main () { int D.1965; int a;
<pre>int *p, *q; void callme (int); int main () { int a, b; p = &b callme (a); return 0; } void callme (int a) {</pre>	<pre>int b; p = &b callme (a); D.1965 = 0; return D.1965; } callme (int a) { int * p.0; int a.1;</pre>
a = *(p + 3); q = &a }	<pre>p.0 = p; a.1 = MEM[(int *)p.0 + 12B] a = a.1; q = &a }</pre>

 	~ ~	
lulv	- 21	
 uiy	- 20	

An Intraprocedural Analysis Application

```
static unsigned int
  intra_gimple_manipulation (void)
  ſ
      basic_block bb;
      gimple_stmt_iterator gsi;
      initialize_var_count ();
      FOR_EACH_BB_FN (bb, cfun)
      ſ
          for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi);
                                                  gsi_next (&gsi))
                analyze_gimple_stmt (gsi_stmt (gsi));
      7
      print_var_count ();
      return 0:
  }
Essential Abstractions in GCC
                                             GCC Resource Center, IIT Bombay
1 July 2011
                    GIMPLE and RTL: Adding a GIMPLE Pass to GCC
              An Intraprocedural Analysis Application
  static unsigned int
```





{

}

1 July 2011

21/45

21/45

An Intraprocedural Analysis Application

21/45

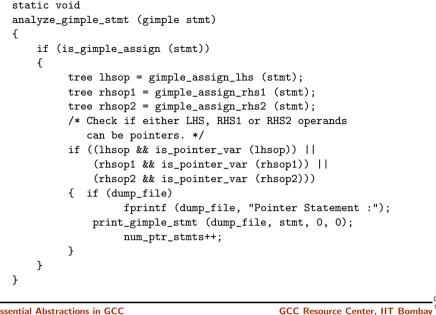
```
static unsigned int
  intra_gimple_manipulation (void)
  ſ
      basic_block bb;
      gimple_stmt_iterator gsi;
      initialize_var_count ();
      FOR_EACH_BB_FN (bb, cfun)
      {
          for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi);
                                                 gsi_next (&gsi))
               analyze_gimple_stmt (gsi_stmt (gsi));
      }
      print_var_count ();
      return 0:
  }
           Basic block iterator parameterized with function
Essential Abstractions in GCC
                                             GCC Resource Center, IIT Bomb
1 July 2011
                    GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                      21/45
               An Intraprocedural Analysis Application
  static unsigned int
  intra_gimple_manipulation (void)
  ſ
      basic_block bb;
      gimple_stmt_iterator gsi;
      initialize_var_count ();
      FOR_EACH_BB_FN (bb, cfun)
      ſ
          for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi);
                                                 gsi_next (&gsi))
               analyze_gimple_stmt (gsi_stmt (gsi));
      }
      print_var_count ()
      return 0:
  }
```

GIMPLE statement iterator



22/45

Analysing GIMPLE Statement



```
Essential Abstractions in GCC
```

```
1 July 2011
                     GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                          22/45
                     Analysing GIMPLE Statement
  static void
  analyze_gimple_stmt (gimple stmt)
  ſ
      if (is_gimple_assign (stmt))
       ſ
            tree lhsop = gimple_assign_lhs (stmt);
            tree rhsop1 = gimple_assign_rhs1 (stmt);
            tree rhsop2 = gimple_assign_rhs2 (stmt);
            /* Check if either LHS, RHS1 or RHS2 operands
               can be pointers. */
            if ((lhsop && is_pointer_var (lhsop)) ||
                (rhsop1 && is_pointer_var (rhsop1)) ||
                (rhsop2 && is_pointer_var (rhsop2)))
            { if (dump_file)
                     fprintf (dump_file, "Pointer Statement :");
                print_gimple_stmt (dump_file, stmt, 0, 0);
                     num_ptr_stmts++;
            }
                              Returns first operand of RHS
       3
  }
```

```
1 July 2011
                     GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                          22/45
                     Analysing GIMPLE Statement
   static void
   analyze_gimple_stmt (gimple stmt)
   Ł
       if (is_gimple_assign (stmt))
       ſ
            tree lhsop = gimple_assign_lhs (stmt);
            tree rhsop1 = gimple_assign_rhs1 (stmt);
            tree rhsop2 = gimple_assign_rhs2 (stmt);
            /* Check if either LHS, RHS1 or RHS2 operands
               can be pointers. */
            if ((lhsop && is_pointer_var (lhsop)) ||
                (rhsop1 && is_pointer_var (rhsop1)) ||
                (rhsop2 && is_pointer_var (rhsop2)))
            { if (dump_file)
                     fprintf (dump_file, "Pointer Statement :");
                print_gimple_stmt (dump_file, stmt, 0, 0);
                     num_ptr_stmts++;
            }
      }
                                     Returns LHS of assignment statement
   }
```

```
Essential Abstractions in GCC
```

GCC Resource Center, IIT Bomb

```
1 July 2011
                     GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                          22/45
                     Analysing GIMPLE Statement
   static void
   analyze_gimple_stmt (gimple stmt)
   ſ
       if (is_gimple_assign (stmt))
       ł
            tree lhsop = gimple_assign_lhs (stmt);
            tree rhsop1 = gimple_assign_rhs1 (stmt);
            tree rhsop2 = gimple_assign_rhs2 (stmt);
            /* Check if either LHS, RHS1 or RHS2 operands
               can be pointers. */
            if ((lhsop && is_pointer_var (lhsop)) ||
                (rhsop1 && is_pointer_var (rhsop1)) ||
                (rhsop2 && is_pointer_var (rhsop2)))
            { if (dump_file)
                     fprintf (dump_file, "Pointer Statement :");
                print_gimple_stmt (dump_file, stmt, 0, 0);
                     num_ptr_stmts++;
            }
                           Returns second operand of RHS
      }
   }
```



ſ

static void

ſ

analyze_gimple_stmt (gimple stmt)

if (is_gimple_assign (stmt))

{ if (dump_file)

}

Essential Abstractions in GCC

static bool

static bool

is_pointer_var (tree var)

is_pointer_type (tree type)

return true;

if (POINTER_TYPE_P (type))

if (TREE_CODE (type) == ARRAY_TYPE)

return AGGREGATE_TYPE_P (type);

}

}

1 July 2011

{

}

{

}

can be pointers. */

num_ptr_stmts++:

Analysing GIMPLE Statement

22/45

1 July 2011

Discovering Pointers

```
static bool
                                                                                          is_pointer_var (tree var)
                                                                                          {
                                                                                              return is_pointer_type (TREE_TYPE (var));
     tree lhsop = gimple_assign_lhs (stmt);
                                                                                          7
     tree rhsop1 = gimple_assign_rhs1 (stmt);
     tree rhsop2 = gimple_assign_rhs2 (stmt);
                                                                                          static bool
     /* Check if either LHS, RHS1 or RHS2 operands
                                                                                          is_pointer_type (tree type)
                                                                                          ſ
     if ((lhsop && is_pointer_var (lhsop)) ||
                                                                                               if (POINTER_TYPE_P (type))
         (rhsop1 && is_pointer_var (rhsop1)) ||
                                                                                                   return true;
         (rhsop2 && is_pointer_var (rhsop2)))
                                                                                               if (TREE_CODE (type) == ARRAY_TYPE)
                                                                                                   return is_pointer_var (TREE_TYPE (type));
              fprintf (dump_file, "Pointer Statement :");
                                                                                               /* Return true if it is an aggregate type. */
         print_gimple_stmt (dump_file, stmt, 0, 0);
                                                                                               return AGGREGATE_TYPE_P (type);
                                                                                          }
                              Pretty print the GIMPLE statement
                                                                                       Essential Abstractions in GCC
                                        GCC Resource Center, IIT Bomba
                                                                                                                                       GCC Resource Center, IIT Bomb
              GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                    23/45
                                                                                       1 July 2011
                                                                                                            GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                                                                                                                  23/45
                    Discovering Pointers
                                                                                                                  Discovering Pointers
                                                                                          static bool
                                                                                          is_pointer_var (tree var)
                                                                                          {
return is_pointer_type (TREE_TYPE (var));
                                                                                              return is_pointer_type (TREE_TYPE (var));
                                                                                          7
                                                                                          static bool
                                                                                          is_pointer_type (tree type)
                                                                                          {
                                                                                               if (POINTER_TYPE_P (type))
                                                                                                   return true;
                                                                                               if (TREE_CODE (type) == ARRAY_TYPE)
     return is_pointer_var (TREE_TYPE (type));
                                                                                                   return is_pointer_var (TREE_TYPE (type));
 /* Return true if it is an aggregate type */
                                                                                               /* Return true if it is an aggregate type. */
                                                                                               return AGGREGATE_TYPE_P (type);
                                                                                          }
```

Defines what kind of node it is

Essential Abstractions in GCC

Data type of the expression

July 2011 GIMPLE and RTL: Adding	a GIMPLE Pass to GCC	24/45	1 July 2011	GIMPLE and RTL: Adding a	GIMPLE Pass to GCC
Intraprocedural Analysis Results			Discovering Local Variables		
			static voi	d gather_local_variables	()
main ()	1		{		-
{			tr	ee list = cfun->local_dee	cls;
p = &b			: -	(Idump file)	
callme (a);	Information collected by in	ntrapro-	11	(!dump_file)	
D.1965 = 0;	cedural Analysis pass			return;	
return D.1965;	• For main: 1		fn	rintf(dump_file,"\nLocal	variables · ").
}			-	ile (list)	Variables .),
callme (int a)	• For callme: 2		{		
1			Ĺ	if (IDECL ARTIFICI)	AL (list) && dump_file)
p.0 = p;	Why is the pointer in the re	d state-		{	(1150) && dump_1110)
a.1 = MEM[(int *)p.0 + 12B];	ment being missed?				<pre>up_file, get_name(list));</pre>
a = a.1;					<pre>up_file,"\n");</pre>
q = &a				}	
}	1			list = TREE_CHAIN (list):
			}		
		~~~~	}		. 1
July 2011 GIMPLE and RTL: Adding		26/45	1 July 2011	GIMPLE and RTL: Adding a	
Discovering Glo	obal Variables		Ad	ding Interprocedural Pa	ss as a Static Plugin
static void gather_global_variables ()			1. Add the following gimple_opt_pass struct instance to the file		
{			struct	simple_ipa_opt_pass pass	s_inter_gimple_manipulation =
<pre>struct varpool_node *node;</pre>			{		
			{		
<pre>if (!dump_file)</pre>				PLE_IPA_PASS,	/* optimization pass type
return;			"gm		
					/* name of the pass*/
<pre>fprintf(dump_file,"\nGlobal variables : ");</pre>			gat	e_gimple_manipulation,	/* gate. */
<pre>for (node = varpool_nodes; node; node = node-&gt;next)</pre>			gat	<pre>e_gimple_manipulation, er_gimple_manipulation,</pre>	/* gate. */ /* execute (driver functions)
{		xt)	gat int NUL	e_gimple_manipulation, er_gimple_manipulation, L,	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */</pre>
<pre>tree var = node-&gt;decl;</pre>			gat int NUL NUL	e_gimple_manipulation, er_gimple_manipulation, L,	<pre>/* gate. */ /* execute (driver function /* sub passes to be run *, /* next pass to run */</pre>
	s; node; node = node->ne •decl;	xt)	gat int NUL NUL 0,	e_gimple_manipulation, er_gimple_manipulation, L,	<pre>/* gate. */ /* execute (driver function /* sub passes to be run *, /* next pass to run */ /* static pass number */</pre>
if (!DECL_ARTIFIC	s; node; node = node->ne •decl;	xt)	gat int NUL NUL O, O,	e_gimple_manipulation, er_gimple_manipulation, L,	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */ /* next pass to run */ /* static pass number */ /* timevar_id */</pre>
if (!DECL_ARTIFIC {	s; node; node = node->ne >decl; CIAL(var))		gat int NUL NUL 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run *, /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */</pre>
if (!DECL_ARTIFIC { fprintf(d	s; node; node = node->ne >decl; CIAL(var)) dump_file, get_name(var)		gat int NUL 0, 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run *, /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */ /* properties provided */</pre>
if (!DECL_ARTIFIC { fprintf(d fprintf(d	s; node; node = node->ne >decl; CIAL(var))		gat int NUL 0, 0, 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */ /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */ /* properties provided */ /* properties destroyed */</pre>
if (!DECL_ARTIFIC { fprintf(d fprintf(d }	s; node; node = node->ne >decl; CIAL(var)) dump_file, get_name(var)		gat int NUL 0, 0, 0, 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */ /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */ /* properties destroyed */ /* todo_flags start */</pre>
<pre>if (!DECL_ARTIFIC {     fprintf(d     fprintf(d } }</pre>	s; node; node = node->ne >decl; CIAL(var)) dump_file, get_name(var)		gat int NUL 0, 0, 0, 0, 0, 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */ /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */ /* properties provided */ /* properties destroyed */</pre>
<pre>if (!DECL_ARTIFIC {     fprintf(d     fprintf(d }</pre>	s; node; node = node->ne >decl; CIAL(var)) dump_file, get_name(var)		gat int NUL 0, 0, 0, 0, 0, 0,	<pre>e_gimple_manipulation, er_gimple_manipulation, L,</pre>	<pre>/* gate. */ /* execute (driver function /* sub passes to be run */ /* next pass to run */ /* static pass number */ /* timevar_id */ /* properties required */ /* properties destroyed */ /* todo_flags start */</pre>

#### Adding Interprocedural Pass as a Static Plugin

# Adding Interprocedural Pass as a Static Plugin

- 2. Write the driver function in file new-pass.c
- Declare your pass in file tree-pass.h: extern struct simple_ipa_opt_pass pass_inter_gimple_manipulation;
- Add your pass to the interprocedural pass list in init_optimization_passes()

```
...
p = &all_regular_ipa_passes;
NEXT_PASS (pass_ipa_whole_program_visibility);
NEXT_PASS (pass_inter_gimple_manipulation);
NEXT_PASS (pass_ipa_cp);
....
```

- 5. In \$SOURCE/gcc/Makefile.in, add new-pass.o to the list of language independent object files. Also, make specific changes to compile new-pass.o from new-pass.c
- 6. Configure and build gcc for cc1
- 7. Debug using ddd/gdb if a need arises (For debuging cc1 from within gcc, see: http://gcc.gnu.org/ml/gcc/2004-03/msg01195.html)

Essential Abstractions in GCC

```
1 July 2011
                    GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                          28/45
             Discovering Pointer Usage Interprocedurally
  static unsigned int
  inter_gimple_manipulation (void)
  ſ
      struct cgraph_node *node;
      basic_block bb;
      gimple_stmt_iterator gsi;
      initialize_var_count ();
      for (node = cgraph_nodes; node; node=node->next) {
         /* Nodes without a body, and clone nodes are not interesting. */
         if (!gimple_has_body_p (node->decl) || node->clone_of)
              continue;
         push_cfun (DECL_STRUCT_FUNCTION (node->decl));
         FOR_EACH_BB (bb) {
             for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi); gsi_next (&gsi))
                  analyze_gimple_stmt (gsi_stmt (gsi));
         }
         pop_cfun ();
      }
      print_var_count ();
      return 0:
  7
```



Essential Abstractions in GCC

GCC Resource Center, IIT Bomba

GCC Resource Center, IIT Bom

```
GIMPLE and RTL: Adding a GIMPLE Pass to GCC
                                                                       28/45
          Discovering Pointer Usage Interprocedurally
static unsigned int
inter_gimple_manipulation (void)
   struct cgraph_node *node;
   basic_block bb;
   gimple_stmt_iterator gsi;
   initialize_var_count ();
   for (node = cgraph_nodes; node; node=node->next) {
      /* Nodes without a body, and clone nodes are not interesting. */
      if (!simple_has_body_p (node->decl) || node->clone_of)
           continue;
      push_cfun (DECL_STRUCT_FUNCTION (node->decl));
      FOR_EACH_BB (bb) {
          for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi); gsi_next (&gsi))
               analyze_gimple_stmt (gsi_stmt (gsi));
      }
      pop_cfun ();
   }
                                        Iterating over all the callgraph nodes
   print_var_count ();
   return 0;
```

GCC Resource Center, IIT Bomba

28/45

1 July 2011

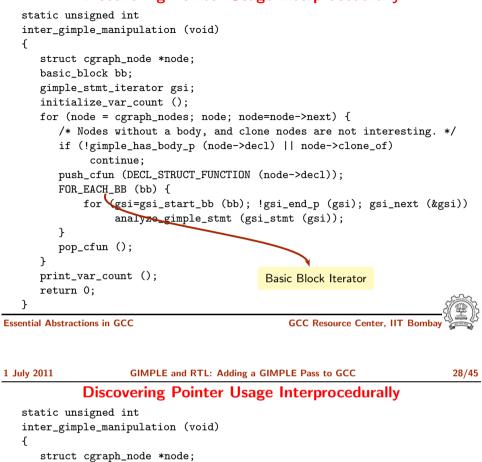
}

#### 28/45

## Discovering Pointer Usage Interprocedurally

**Discovering Pointer Usage Interprocedurally** static unsigned int inter_gimple_manipulation (void) ł struct cgraph_node *node; basic_block bb; gimple_stmt_iterator gsi; initialize var count (); for (node = cgraph_nodes; node; node=node->next) { /* Nodes without a body, and clone nodes are not interesting. */ if (!gimple_has_body_p (node->decl) || node->clone_of) continue: push_cfun (DECL_STRUCT_FUNCTION (node->decl)); FOR_EACH_BB (bb) { for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi); gsi_next (&gsi)) analyze_gimple_stmt (gsi_stmt (gsi)); } pop_cfun (); } Setting the current function in context print_var_count (); return 0; } Essential Abstractions in GCC GCC Resource Center, IIT Bomba 1 July 2011 GIMPLE and RTL: Adding a GIMPLE Pass to GCC 28/45 **Discovering Pointer Usage Interprocedurally** static unsigned int inter_gimple_manipulation (void) ſ struct cgraph_node *node; basic_block bb; gimple_stmt_iterator gsi; initialize_var_count (); for (node = cgraph_nodes; node; node=node->next) { /* Nodes without a body, and clone nodes are not interesting. */ if (!gimple_has_body_p (node->decl) || node->clone_of) continue; push_cfun (DECL_STRUCT_FUNCTION (node->decl)); FOR_EACH_BB (bb) { for (gsi=gsi_start_bb (bb); !gsi_end_p (gsi); gsi_next (&gsi)) analyze_gimple_stmt (gsi_stmt (gsi)); } pop_cfun (); } print_var_count (); **GIMPLE** Statement Iterator return 0: 7 **Essential Abstractions in GCC** 

GCC Resource Center, IIT Bomba



Discovering Pointer Usage int	terprocedurally
static unsigned int	
inter_gimple_manipulation (void)	
{	
<pre>struct cgraph_node *node;</pre>	
<pre>basic_block bb;</pre>	
<pre>gimple_stmt_iterator gsi;</pre>	
<pre>initialize_var_count ();</pre>	
<pre>for (node = cgraph_nodes; node; node=node-</pre>	->next) {
/* Nodes without a body, and clone node	es are not interesting. */
if (!gimple_has_body_p (node->decl)	node->clone_of)
continue;	
push_cfun (DECL_STRUCT_FUNCTION (node->	>decl));
FOR_EACH_BB (bb) {	
<pre>for (gsi=gsi_start_bb (bb); !gsi_en</pre>	nd_p (gsi); gsi_next (&gsi))
analyze_gimple_stmt (gsi_stmt	(gsi));
}	
<pre>pop_cfun ();</pre>	
}	
print_var_count (); Resett	ing the function context
return 0;	-7-
}	S CONTRACTOR OF

GIMPLE and RTL: Adding a GIMPLE Pass to GCC
Interprocedural Results

29/45

Number of Pointer Statements = 3

#### Observation:

- Information can be collected for all the functions in a single pass
- Better scope for optimizations

Part 4 An Overview of RTL



Assembly language for an abstract machine with infinite registers

Why RTL?

#### A lot of work in the back-end depends on RTL. Like,

• Low level optimizations like loop optimization, loop dependence, common subexpression elimination, etc

GIMPLE and RTL: An Overview of RTL

- Instruction scheduling
- Register Allocation
- Register Movement





GIMPLE and RTL: An Overview of RTL

For tasks such as those, RTL supports many low level features, like,

Why RTL?

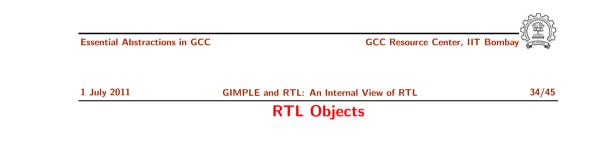
32/45

1 July 2011

GIMPLE and RTL: An Overview of RTL The Dual Role of RTL

- For specifying machine descriptions Machine description constructs:
  - define_insn, define_expand, match_operand
- For representing program during compilation IR constructs
  - insn, jump_insn, code_label, note, barrier

This lecture focusses on RTL as an IR



- Types of RTL Objects
  - Expressions
  - Integers
  - Wide Integers
  - Strings

**Essential Abstractions in GCC** 

- Vectors
- Internal representation of RTL Expressions
  - ► Expressions in RTX are represented as trees
  - $\blacktriangleright$  A pointer to the C data structure for RTL is called <code>rtx</code>

Essential Abstractions in GCC

• Register classes

• Memory addressing modes

• Compare and branch instructions

• Word sizes and types

Calling ConventionsBitfield operations

Part 5

An Internal View of RTL



GCC Resource Center, IIT



GIMPLE and RTL: An Internal View of RTL

**RTX Codes** 

38/45

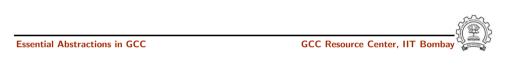
**RTL Classes** 

RTL Expressions are classified into RTX codes :

- Expression codes are names defined in rtl.def
- RTX codes are C enumeration constants
- Expression codes and their meanings are machine-independent
- Extract the code of a RTX with the macro  $\texttt{GET_CODE}(\mathtt{x})$

RTL expressions are divided into few classes, like:

- RTX_UNARY : NEG, NOT, ABS
- RTX_BIN_ARITH : MINUS, DIV
- RTX_COMM_ARITH : PLUS, MULT
- RTX_OBJ : REG, MEM, SYMBOL_REF
- RTX_COMPARE : GE, LT
- RTX_TERNARY : IF_THEN_ELSE
- RTX_INSN : INSN, JUMP_INSN, CALL_INSN
- RTX_EXTRA : SET, USE



1 July 2011	GIMPLE and RTL: An Internal View of RTL
	RTX Codes

The RTX codes are defined in rtl.def using cpp macro call DEF_RTL_EXPR, like :

- DEF_RTL_EXPR(INSN, "insn", "iuuBieie", RTX_INSN)
- DEF_RTL_EXPR(SET, "set", "ee", RTX_EXTRA)
- DEF_RTL_EXPR(PLUS, "plus", "ee", RTX_COMM_ARITH)
- DEF_RTL_EXPR(IF_THEN_ELSE, "if_then_else", "eee", RTX_TERNARY)

The operands of the macro are :

- Internal name of the rtx used in C source. It's a tag in enumeration enum rtx_code
- $\bullet\,$  name of the <code>rtx</code> in the external ASCII format
- Format string of the rtx, defined in rtx_format[]
- $\bullet$  Class of the <code>rtx</code>



GCC Resource Center, IIT

1 July 2011 GIMPLE and RTL: An Internal View of RTL
RTX Formats

DEF_RTL_EXPR(INSN, "insn", "iuuBieie", RTX_INSN)

• i : Integer

Essential Abstractions in GCC

- $\bullet\ u$  : Integer representing a pointer
- B : Pointer to basic block
- $\bullet \ e \ : \ \mathsf{Expression}$



## Basic RTL APIs

- XEXP,XINT,XWINT,XSTR
  - Example: XINT(x,2) accesses the 2nd operand of rtx x as an integer
  - Example: XEXP(x,2) accesses the same operand as an expression
- Any operand can be accessed as any type of RTX object
  - So operand accessor to be chosen based on the format string of the containing expression
- Special macros are available for Vector operands
  - XVEC(exp,idx) : Access the vector-pointer which is operand number idx in exp
  - XVECLEN (exp, idx ) : Access the length (number of elements) in the vector which is in operand number idx in exp. This value is an int
  - XVECEXP (exp, idx, eltnum): Access element number "eltnum" in the vector which is in operand number idx in exp. This value is an RTX

Essential Abstractions in GCC

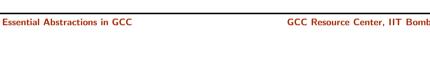
GCC Resource Center, IIT Bombay

Part 6

Manipulating RTL IR







GIMPLE and RTL: An Internal View of RTL

**RTL** statements

1 July 2011

1 July 2011

GIMPLE and RTL: An Internal View of RTL RTL Insns

- A function's code is a doubly linked chain of INSN objects
- Insns are rtxs with special code
- Each insn contains atleast 3 extra fields :
  - Unique id of the insn , accessed by INSN_UID(i)
  - PREV_INSN(i) accesses the chain pointer to the INSN preceeding i
  - NEXT_INSN(i) accesses the chain pointer to the INSN succeeding i
- The first insn is accessed by using get_insns()
- The last insn is accessed by using get_last_insn()

-----

• RTL insns contain embedded links

CALL_INSN : Function calls

► BARRIER : End of control Flow

▶ NOTE : Debugging information

• Types of RTL insns :

• RTL statements are instances of type rtx

INSN : Normal non-jumping instruction

CODE_LABEL: Target label for JUMP_INSN

JUMP_INSN : Conditional and unconditional jumps



GIMPLE and RTL: Manipulating RTL IR

Adding an RTL Pass

42/45

1 July 2011

43/45

45/45

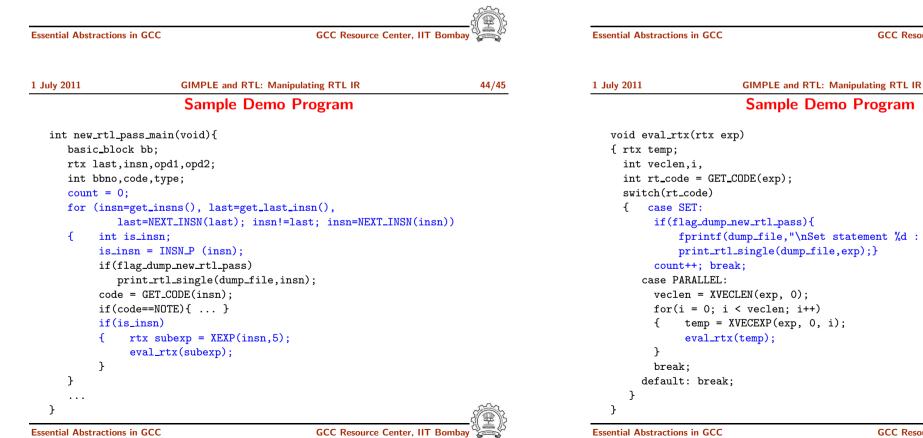
## Sample Demo Program

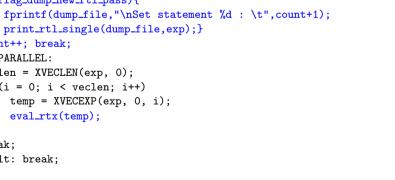
Similar to adding GIMPLE intraporcedural pass except for the following

- Use the data structure struct rtl_opt_pass
- Replace the first field GIMPLE_PASS by RTL_PASS

Problem statement : Counting the number of SET objects in a basic block by adding a new RTL pass

- Add your new pass after pass_expand
- new_rtl_pass_main is the main function of the pass
- Iterate through different instructions in the doubly linked list of instructions and for each expression, call eval_rtx(insn) for that expression which recurse in the expression tree to find the set statements





GCC Resource Center, IIT

GCC Resource Center, IIT